PROYECTO FIN DE CARRERA

CURSO 2011-2012

SOCIALSPORT

Red social deportiva para Android

Autores:

Luis Cruz Bellas

José Rodríguez de Llera Tejeda

Álvaro Zapata Montes

Directora:

María Victoria López López

Este trabajo se lo dedicamos

a nuestra familia más cercana.

AUTORIZACIÓN

Luis Cruz Bellas, José Rodríguez de Llera Tejeda y Álvaro Zapata Montes, alumnos matriculados en la asignatura de Sistemas Informáticos, autorizan, mediante el presente documento, a la Universidad Complutense de Madrid (UCM), a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y el prototipo desarrollado, todo ello realizado durante el curso académico 2011-2012 bajo la dirección de María Victoria López López, profesora del Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática de dicho organismo.

Luis Cruz Bellas José Rodríguez de Llera Tejeda Álvaro Zapata Montes

AGRADECIMIENTOS

Agradecemos a nuestras familias por habernos apoyado a lo largo de toda la carrera, cuyo final está cada vez más cerca y culmina con la presentación del proyecto de Sistemas Informáticos.

No sería justo olvidarnos de nuestra directora de proyecto, María Victoria López López, no sólo por haber acogido nuestra propuesta con los brazos abiertos, sino por prestarnos su ayuda durante todo el proceso y haber aportado su granito de arena.

PRÓLOGO

Este proyecto denominado SocialSport es una idea original de los autores de esta memoria: Álvaro, Luis y José. Ellos propusieron el desarrollo de una red social deportiva que resolviera los problemas que como deportistas sufrían cada día. Los tres son unos fantásticos desarrolladores y forman un equipo excepcional tanto en el trabajo como en su tiempo libre. Todos ellos son amantes del deporte hasta el punto de llevar su formación y su proyecto a esta área. Su entusiasmo por la red social deportiva que han desarrollado ha sido la parte fundamental del éxito de este proyecto. Todos ellos se han involucrado al máximo invirtiendo tantas horas de estudio y trabajo como han podido encontrar a lo largo del curso y se han empeñado en realizar todas las propuestas por mi parte. Por las razones expuestas, este ha sido uno de los proyectos más fácil de dirigir por mi parte. En cualquier tipo de trabajo, el entusiasmo y la motivación siempre es la clave del éxito y en este caso estos componentes no han faltado en ningún momento.

Quiero destacar la buena disposición de Luis, José y Álvaro ante la propuesta del por parte del Grupo G-TeC para integrar su herramienta en el proyecto JHoy!, proyecto para la reserva de pistas en instalaciones deportivas universitarias, y para lo que tuvieron que desarrollar un prototipo de simulación basado en los datos de las instalaciones de la Zona Sur y Paraninfo de la UCM. También quiero destacar que la aplicación final SocialSport ha sido presentada a los Servicios Informáticos de la UCM y a dos vicerrectorados involucrados en la integración de la herramienta y que en ambos casos la UCM ha manifestado tanto su interés como su aprobación al desarrollo realizado por los autores de este proyecto. Desde aquí les deseo mucha suerte y les agradezco haber confiado en mí para dirigir este magnífico proyecto.

Victoria López

RESUMEN

El objetivo de este proyecto denominado SocialSport es el desarrollo de una red social enfocada al mundo del deporte, cuyo objetivo principal consiste en la organización de eventos deportivos, para dispositivos móviles que utilicen el sistema operativo Android. La funcionalidad de la aplicación se completa con la posibilidad de interactuar y comunicarse con otros usuarios, formar parte de diferentes grupos y equipos, recibir avisos en tiempo real de las novedades, gestionar el perfil, incluir nuevas localizaciones y muchas otras opciones.

Además, incorpora un simulador de reservas sobre el Paraninfo Norte y Paraninfo Sur, instalaciones de la Universidad Complutense de Madrid, teniendo en cuenta horarios, deportes y precios reales, a través de un algoritmo de búsqueda, en función de unos parámetros dados por un usuario. La finalidad del simulador es sentar las bases de un futuro sistema de reservas real para universidades y complejos deportivos.

Palabras clave: Android, aplicaciones móviles, redes sociales, eventos deportivos, deporte.

ABSTRACT

The objective of this project called SocialSport is the development of a social network focused on the sports world, whose main objective is organizing sporting events, for mobile devices using the Android operating system. The functionality of the application is completed with the ability to interact and communicate with other users, join groups and teams, receive real-time news alerts, manage your profile, add new locations and many other options.

It also incorporates a simulator of reservations over Paraninfo Norte and Paraninfo Sur, both facilities of the Universidad Complutense of Madrid, taking into account schedules, sports and real prices, through a search algorithm, based on parameters given by a user. The purpose of the simulator is to lay the foundations for a future real-booking system for universities and sporting complexes.

Keywords: Android, mobile applications, social networking, sporting event, sport.

ÍNDICE

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

Presentación

SocialSport es la unión de redes sociales, deporte y aplicaciones para dispositivos móviles. Este concepto era nuevo cuando se empezó a desarrollar esta memoria. Existían métodos de organizar eventos deportivos, existían redes sociales y existían aplicaciones para dispositivos móviles, pero no la conjunción de todas. Ahora, esas empresas que se dedicaban a organizar eventos se están preocupando por el mercado existente en el mundo de los Smartphones, y ya han publicado sus primeras versiones de este tipo de aplicaciones.

Organización de la memoria

Esta memoria comienza con una breve explicación sobre qué es SocialSport y cómo se gestó la idea de desarrollar este software para dispositivos Android. También se incorpora un breve estudio sobre las aplicaciones líderes del mercado, reseñando sus características y aspectos más importantes. A continuación, se exponen todos los aspectos relativos al desarrollo de SocialSport. Los requisitos y recursos necesitados, las herramientas empleadas, la estructura de la aplicación, además de todos los problemas surgidos, con sus correspondientes soluciones. Para terminar, se incluye un manual de usuario en el que se explican, detalladamente, todas y cada una de las funcionalidades disponibles.

Como anexo, se incorpora un manual de Android que permite el acercamiento al conocimiento de este sistema operativo y al desarrollo de aplicaciones, a través del entorno de desarrollo Eclipse, para Android.

Motivación

Estamos dentro de un grupo de amigos que habitualmente practica diferentes deportes. Porque nos gusta. Porque nos divierte. Y además es sano. Lo extraño es que siendo personas con características tan similares resulte tan difícil ponerse de acuerdo para jugar cualquier tipo de partido. Que si problemas con la hora, problemas con el lugar, bajas de última hora, etcétera. Todo pegas. Al que se le ocurre organizarlo sabe que lo más importante de todo es la paciencia, pues el proceso no será ni corto ni sencillo.

De ahí surgió la idea inicial de SocialSport (Figura 1). Habíamos vivido esta situación

demasiadas veces, y teníamos claro que no queríamos que se repitiera. Se debía hacer algo. El objetivo era poder practicar el deporte que te apeteciera, cuando te apeteciera, donde te apeteciera y, sobre todo, no tener que vivir pendiente de cientos de conversaciones, llamadas y otras pérdidas de tiempo.

Figura 1. Título y eslogan de SocialSport

La solución pasaba por un dispositivo móvil. Es la única forma de solucionar el "cuándo quiera". Casi todo el mundo pasa el día con su teléfono en el bolsillo, a su lado, encima de la mesa, etcétera. Siempre cerca. Es prácticamente la definición de información en tiempo real. Si me apetece jugar un partido después del trabajo, sin pasar por casa y sin perder demasiado tiempo, es mi móvil quien me lo tiene que conseguir, ya sea como creador o como invitado.

Con la idea básica en mente, y la plataforma establecida, sólo quedaba desarrollar un software que permitiera llevar a cabo nuestras intenciones. Entonces surgió un nombre: Android (para más información, consultar anexo). Todo lo relacionado con este sistema operativo es gratuito, y se pueden crear aplicaciones a través de un entorno conocido para nosotros, como es Eclipse, y en un lenguaje sencillo, como JAVA. Esta conjunción nos facilitaba su desarrollo.

El problema inicial que hemos encontrado ha sido el tiempo. Los estudios, los equipos, las parejas, los trabajos, etcétera. Todo requiere mucha dedicación y esfuerzo, así que la decisión fue clara: proponer el proyecto como tema de la asignatura Sistemas Informáticos. Introducirlo en algo a lo que íbamos a dedicar tiempo, y que nos permitiera llevar a cabo esta idea y aprender a la vez. La tarea fue ardua, pero tras la recomendación de un amigo, encontramos a la persona adecuada, que no sólo nos aceptó la propuesta, sino que participó activamente con nuevas ideas y recomendaciones y sugerencias.

Conceptos

En la sociedad tan avanzada en la que vivimos, todavía resulta muy complicado ponerse de acuerdo con unos cuantos amigos para practicar casi cualquier deporte colectivo. Siempre falta alguno, o a última hora se produce una baja. Las llamadas y conversaciones van de arriba abajo sin llegar a conclusión alguna. Hay que remover cielo y tierra para que el asunto salga adelante. Ponerse de acuerdo es casi una utopía. Con lo fácil que es decir sí o no, lo complicado que puede llegar a resultar.

Por otro lado, hoy en día es poco habitual que alguien no tenga un dispositivo móvil. Es más, la mayoría de estos terminales incorporan acceso a Internet. Se puede concluir que estamos en la época de los dispositivos móviles, y que a diario vivimos con al menos uno de estos aparatos pegado a nosotros.

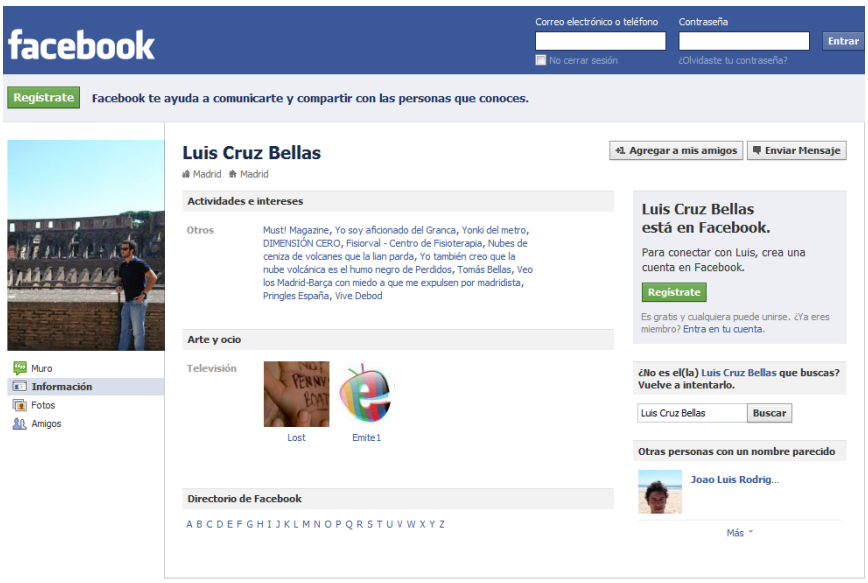


Figura 2. Perfil público de Facebook

Además, ser miembro de una red social (Figura 2) es casi una necesidad. Hasta tal punto que las utilizamos para informarnos e informar, para conocer gente y que nos conozcan, e incluso para publicar nuestro perfil y currículum laboral de cara a obtener nuevas posibilidades de trabajo.

El concepto que vamos a desarrollar a lo largo de esta memoria está muy relacionado con los aspectos mencionados en esta sección. Es, de hecho, la unión de 3 partes: aplicaciones móviles, redes sociales y organización de eventos deportivos.

Redes Sociales

Las redes sociales son, principalmente, una forma de relación entre individuos conectados por amistad, un mismo entorno, parentesco, etcétera. Son una "reunión virtual". Y se han convertido en un medio de comunicación e información. Prácticamente en una forma de vida. Permiten compartir vídeos, fotos, vivencias, opiniones, incluso encontrar personas con quien llevas tiempo sin hablar.

Figura 3. Perfil público de Twitter

Actualmente, su expansión y búsqueda de continuas mejoras parece imparable. Son el centro de atención de Internet. Algunas de ellas ofrecen múltiples posibilidades y otras se centran más en 1 o 2 aspectos. Por ejemplo, Twitter (Figura 3) se basa más en los comentarios de la gente, mientras que Facebook, Google+ y otras redes sociales son más generales y destacan por la cantidad de imágenes y vídeos que se comparten.

Aplicaciones móviles

Una aplicación móvil es la implementación informática de un programa destinado a ser utilizado en un dispositivo portátil. Para que sea exitosa, es preciso que sea sencilla de manejar, funcional, llamativa visualmente, y que sea popular. De hecho, quizá la última característica sea la más importante, pero es difícil que se cumpla si no se tienen en

cuenta las anteriores.

La distribución de este tipo de aplicaciones es cada vez más amplia, y podemos encontrar una diversidad de contenidos y posibilidades casi infinita. Y lo que es más importante: cada vez son más útiles y facilitan más la vida cotidiana. Los lugares más destacados desde donde se pueden obtener son Google Play (hasta hace poco Android Market, Figura 4) y App Store, siendo, cada uno de estos mercados, líderes para su sistema operativo correspondiente. Ahí, las aplicaciones se organizan por múltiples categorías (juegos, comunicación, transportes, tiempo…), e incluso subcategorías (juegos de acción, de deporte…), pudiendo ser gratuitas o no.

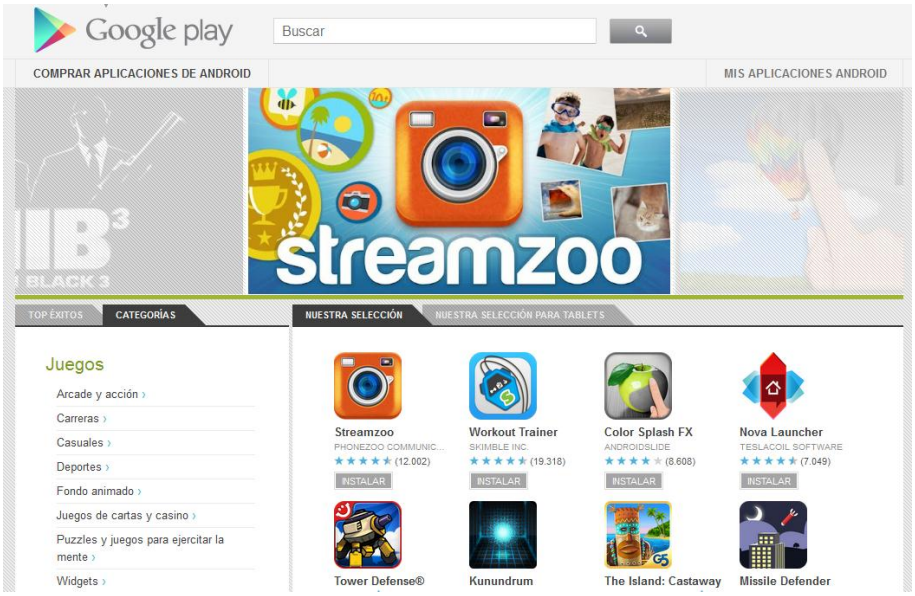


Figura 4. Google Play (Market de Android)

Organización de eventos deportivos

La organización de eventos deportivos no es un concepto innovador en sí mismo. Lleva muchos años llevándose a cabo de casi todas las formas posibles, y a todos los niveles. El principal problema es encontrar participantes, recintos, horarios flexibles de modo que satisfagan a casi todos y, a determinado nivel, patrocinadores. Una forma de incentivar a nuevos candidatos a inscribirse en determinado evento es recompensar al ganador, lo que también suele suponer un aumento del nivel de competitividad.

Por otro lado, aquellos que se dediquen a organizarlo saben que no va a resultar nada sencillo. Hay medios que facilitan la labor (Figura 5), pero pocos están diseñados ex profeso. Normalmente se anuncian en zonas o lugares cuyo entorno es más proclive a la participación en ellos, lo que aumenta la probabilidad de encontrar participantes y,

por ende, de que salga adelante.

Figura 5. Ejemplo de web organizadora de eventos deportivos

ESTADO DEL ARTE

SocialSport es una aplicación móvil a través de la cual pueden organizarse eventos deportivos, facilitando la búsqueda de participantes y recintos donde llevarse a cabo, mostrando toda la información necesaria, y permitiendo la comunicación entre participantes. Además, también tiene un perfil social mediante el cual puedes conocer nueva gente, nuevos equipos y, como no, mantener el contacto con tu entorno, ya sea de forma individual (amigos) o colectiva (grupos, equipos). A continuación, analizaremos algunas aplicaciones similares existentes.

Timpik

Se trata de la red social deportiva más utilizada a nivel nacional. En las figuras 6, 7, 8 y 9 se muestra la pantalla inicial, el perfil de usuario, la pizarra y el acceso de la aplicación de Timpik para Android.

Figura 6. Portada de Timpik

Figura 7. Perfil de usuario registrado de Timpik

Figura 8. Muro de Timpik

Figura 9. Acceso a Timpik

Historia y opiniones

Timpik como web surge en 2009, bajo unas circunstancias similares a las que nace este proyecto, y con el nombre "Falta1". Un pequeño grupo de amigos jiennense que se dedican a la informática, entre los que se encuentra Camilo López de Felipe, el creador, tienen problemas para practicar sus deportes favoritos, así que deciden poner solución al asunto creando este sitio que les facilite la labor.

Como aplicación para dispositivos móviles, aparece bien avanzado el año 2011. Inicialmente, sólo estaba disponible para iPhone, y como es lógico, su funcionamiento era muy mejorable. La versión de salida fue la 1.0 y ahora se ha publicado la 1.6, que principalmente soluciona problemas que había en la anterior.

La versión Android se hizo esperar algo más, y no apareció hasta finales de 2011. Hoy en día, la versión disponible es la 1.49, habiendo sido publicadas más de 10 actualizaciones, desde la 1.0 inicial. En la última mejora, se incluye la solución de algunos problemas técnicos y la adición de funcionalidades no disponibles hasta ese momento.

Actualmente, la versión para Android y la de iPhone están a la par en cuanto a desarrollo. Tanto sus ventajas como inconvenientes se han ido exponiendo a lo largo

de este apartado de la memoria, pero la aplicación sigue evolucionando y se está trabajando cada día para mejorarla.

La aceptación que ha tenido hasta el momento es muy buena. Al tratarse de una idea novedosa, mucha gente la encuentra muy útil y se da cuenta de que esos problemas que han tenido para organizar eventos deportivos pueden solucionarse utilizándola. Por otro lado, el conseguir un mayor número de usuarios registrados (actualmente hay más de 30.000) sería fundamental para que la aplicación fuera mejorando, puesto que al fin y al cabo son ellos los que tienen problemas y ocurrencias de nuevas ideas que, al comunicarlas al soporte técnico, la hacen mejorar.

Figura 10. Configuración de perfil de Timpik

Figura 11. Buscador de eventos de Timpik

Pros y contras de la funcionalidad

Las características y funcionalidades que tiene esta aplicación son muchas. Organización de eventos deportivos, con múltiples opciones de cara a que los participantes no tengan ninguna duda de la naturaleza del mismo; perfil de usuario muy completo, que incluye detalles interesantes como puede ser el número de teléfono, la fiabilidad, su nivel y comportamiento en los partidos en los que ha participado (Figura 10); perfil social; buscador de partidos (Figura 11); posibilidad de

añadir nuevos campos a la base de datos y otras características, son sólo unos ejemplos.

Por otro lado, se echan de menos determinados aspectos: perfil grupal, (los miembros de un equipo no pueden acceder a los contenidos desde la aplicación); árbitros (muchas veces necesarios); organización de torneos (probablemente por su complejidad); o permitir obtener tu ubicación por la conexión a internet, lo cual, como veremos más adelante, puede suponer un contratiempo importante. Tampoco permite añadir gente ajena a la aplicación.

Características que la hacen interesante en el mercado

En estos momentos, estamos sin duda ante la aplicación líder en el mercado de estas características. Su manejo es sencillo e intuitivo, y su utilidad es novedosa. La repercusión a nivel nacional es cada vez mayor e incluso ha llegado a ser noticia en la televisión. También en Latinoamérica tiene su mercado. De momento no puede competir en volumen de usuarios con grandes redes sociales como Tuenti y menos aún con Facebook (con quién está relacionada y puedes utilizar tu usuario para acceder) o Twitter, pero es la número 1 de su sector en este aspecto. De hecho, consta de inversores que subvencionan su crecimiento.

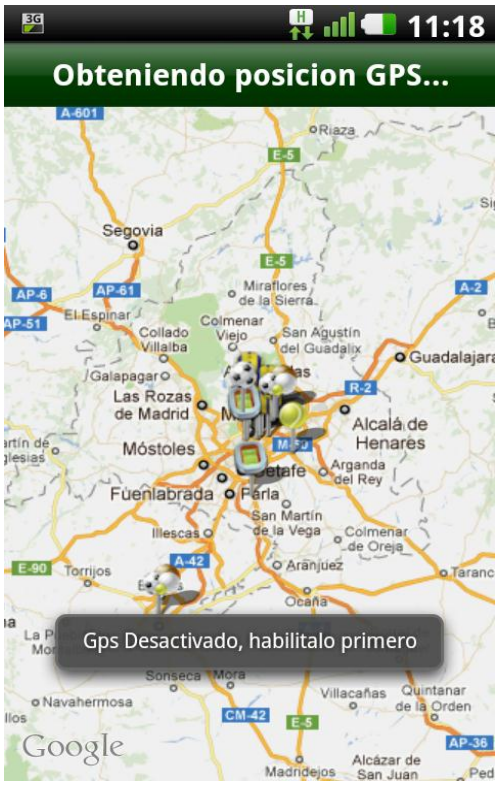


Figura 12. Obtención ubicación Timpik (sólo GPS)

Figura 13. Recomendador de partidos

Crear un nuevo evento es muy fácil y cómodo, y te permite configurar múltiples

opciones sin perder casi tiempo. Además, el usuario puede recibir avisos, tanto en el propio teléfono como por mail, de las novedades. Todo esto supone su mayor logro, puesto algo así no existía en el mercado. Y un acierto es querer hacer lo básico con calidad, antes que abarcar mucho y hacerlo de cualquier manera.

La búsqueda manual de eventos deportivos es muy completa. Ofrece muchas posibilidades, y el usuario puede realizarla a su antojo. Basándose en la localización, en la fecha, en la ubicación actual del usuario (como hemos mencionado anteriormente, sólo se obtiene mediante GPS – Figura 12 -, y a veces deseas buscar un evento cercano mientras estás dentro de un edificio, con lo que la señal no se podrá activar), y otras opciones.

Mejoras posibles

A pesar de su posicionamiento en el mercado, se echan en falta algunos aspectos de mejora. La posibilidad de incluir árbitros en los partidos es uno de ellos. En determinados deportes, sobre todo en encuentros de alta competitividad, a menudo es necesaria una figura imparcial que se encargue de controlar el juego. Incluso facilitaría, con su alegato, la evaluación y comprensión de sucesos fuera de lo normal, como pueden ser discusiones, abandonos, incumplimiento de normas, etcétera.

Figura 14. Selección de deportes de Timpik

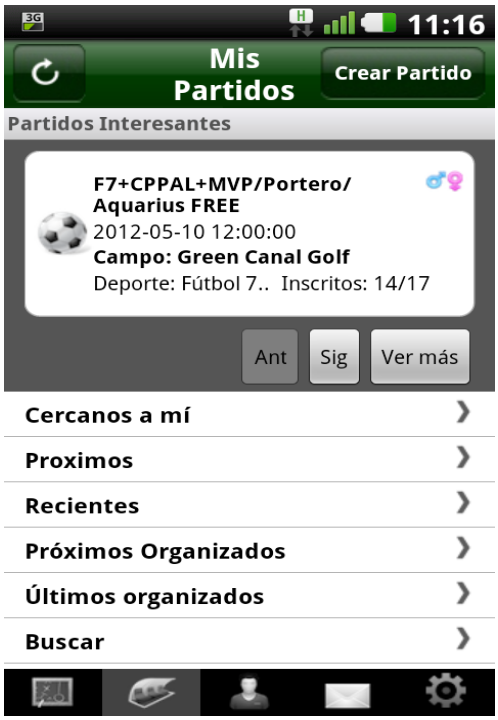


Figura 15. Menú partidos Timpik

Además, desde la aplicación móvil, no es posible organizar torneos ni acceder a ellos.

Es más, en la versión web, tampoco se puede directamente, permitiendo sólo la muestra de información al respecto, y un enlace o teléfono de referencia. Y todo ello tras contactar previamente con miembros del equipo técnico de Timpik. En este caso, somos comprensivos y entendemos que incluir esta funcionalidad no es sencillo, entre otras cosas porque en muchos casos la fiabilidad del organizador es nula y restaría "seriedad" general.

Otro aspecto mejorable son las recomendaciones de eventos. Los criterios a tener en cuenta deberían ser, fundamentalmente, los deportes que el usuario tenga seleccionados en su perfil, y su lugar activo de localización, de modo que se listen, por orden de cercanía, sólo aquellos cuya disciplina es practicada por el usuario. Actualmente no es así, y el recomendador (figuras 13, 14 y 15) muestra eventos de todo tipo de deportes.

Figura 16. Presentación Timpik

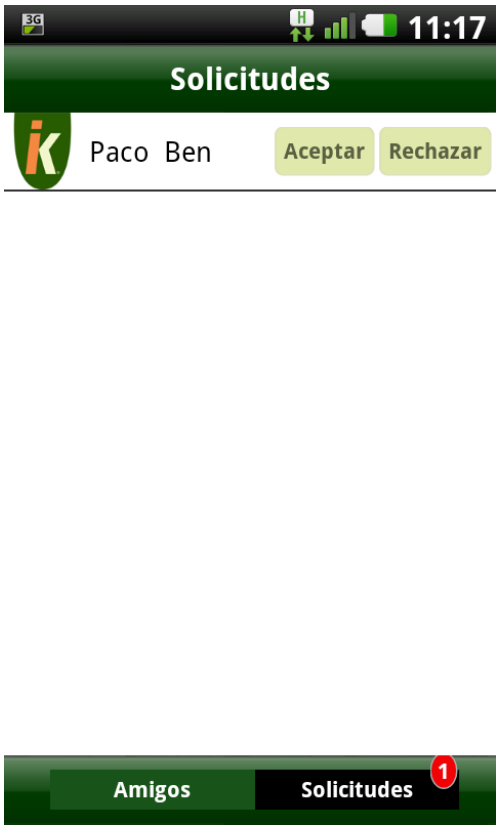


Figura 17. Solicitudes Timpik

A nivel técnico, principalmente hay 2 aspectos que deben ser mejorados, y que no dudamos se hará en un futuro, si se sigue desarrollando la aplicación. El primero es el aspecto general, que no es desagradable, pero tampoco llama la atención. En algunas opciones es incluso rústico y extremadamente simple, en otras es muy llamativo, y en

ocasiones es simple, indiferente para el usuario (figuras 16 y 17).

El otro aspecto técnico mejorable, es el trato que hay con las vistas, tanto a la hora de acceder, como a la de salir de la aplicación, sin haberse realizado la desconexión de sesión. En el primer caso, es preciso acceder por pantallas innecesarias e incluso pulsar algún botón. Además, se refresca de forma automática, pero se produce un efecto visual desagradable para el usuario. Para salir de la aplicación, es necesario pasar de nuevo por las pantallas de registro, y se hace pesado al tener que realizar múltiples pulsaciones para cerrarla.

Sería una gran ventaja poder hacer reservas de pistas cuyo uso y disfrute no es gratuito. Comprendemos que no es algo sencillo, pues habría que establecer acuerdos con polideportivos, ayuntamientos y otros organismos, pero el nivel de comodidad que se alcanzaría sería excelso.

Sitio web

De la versión web de Timpik realizaremos una muy breve reseña, puesto que no es menester en esta memoria. Las conclusiones son similares respecto a la versión para dispositivos móviles. El aspecto general, sin ser malo, es mejorable, pero la funcionalidad es muy completa (incluye posibilidades que no están implementadas en la aplicación), y es muy sencilla de utilizar.

Enlace al sitio web de Timpik

Fubles

Permite crear partidos con múltiples opciones, en ese aspecto es una aplicación muy completa. También está muy conseguido el tema de la fiabilidad de los usuarios a la hora de acudir/organizar un evento, y la evaluación técnica de los mismos, es decir, al medir el nivel de los jugadores.

Historia y opiniones

Fubles nace en 2006 en Italia en su versión web. Sin embargo, no aparece en App Store hasta 2011, y hasta el momento sólo ha habido 1 actualización. La última versión disponible fue lanzada el 8 de julio de 2011, y es sobre la que hemos hecho nuestro análisis (en las figuras 18 y 19, se muestran las pantallas de inicio y partidos respectivamente). Aunque la versión web sigue desarrollándose y mejorando cada día, no sucede lo mismo con la aplicación, que parece estancada y sin visos de que se

publique una actualización en un breve periodo de tiempo.

La repercusión a nivel nacional de Fubles no es muy grande. El concepto o idea es bueno y aceptado por el público al que va dirigido, pero como aplicación aún tiene mucho margen de mejora. Se puede dar por hecho que a mayor demanda, más rápidamente mejoraría el producto, y mejor valoración general obtendría. No obstante, en el país de origen de esta aplicación, Italia, la repercusión es mayor, y puede darse la circunstancia de que los usuarios de un lugar (o globalmente) vean constantes mejoras debido a la demanda existente en otra situación geográfica.

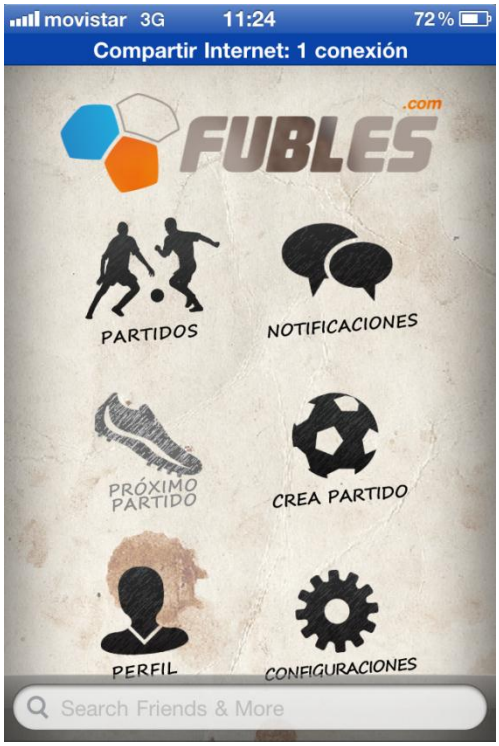


Figura 18. Inicio Fubles

Figura 19. Menú partidos Fubles

Pros y contras de la funcionalidad

Por otro lado, la carencia más grande es que sólo está disponible para iOS, en una época en la que el crecimiento de Android es más que evidente y ha irrumpido en el mercado Windows Phone. Al igual que Timpik, carece de figuras arbitrales y no permite que un usuario incluya en un partido gente que no esté registrada en la aplicación.

Características que la hacen interesante en el mercado

La creación de partidos se basa en la localización donde el usuario quiere jugar, por lo

que la disponibilidad de deportes se basará en las condiciones del lugar seleccionado (Figura 23). Esto evita posibles confusiones y a la vez establece un control necesario para este tipo de aplicaciones. Así resulta imposible planear un partido de baloncesto donde sólo haya instalaciones de tenis, por ejemplo, ya sea por error o adrede.

La interfaz es bastante atractiva, sin llegar a parecer recargada, lo que supone un detalle importante (Figura 20). También es completa, es decir, los elementos existentes no sobran, y no se echa en falta prácticamente nada. Además, su uso es bastante intuitivo por lo que no cuesta encontrar las diferentes opciones (Figura 21).

Mejoras posibles

La primera mejora a tener en cuenta es la traducción al castellano, que en este momento es bastante pobre. Al sacar una aplicación en un mercado extranjero con el que no se comparta idioma, es importante cuidar el lenguaje y vocabulario, puesto que de no ser así se empobrece la imagen y la valoración general.

Figura 20. Estadísticas equipos Fubles

Figura 21. Información partidos Fubles

Por otro lado, Fubles presenta algunos problemas graves: pide una fotografía del perfil en cada conexión al sistema. Es más, en determinadas ocasiones el sistema se bloquea,

probablemente, por cuestiones de servidor. No obstante, no parece que Fubles esté desarrollando nuevas versiones de la aplicación.

Tampoco destaca positivamente el recomendador de eventos (Figura 23). La raíz del problema reside en que en ningún momento ni lugar el usuario puede introducir sus preferencias deportivas. Tampoco existe un buscador, por lo que el problema se agrava. Incluso se muestra un contador de partidos interesantes, accedes a ellos y el número no corresponde con los que realmente hay.

Figura 22. Seleccionar lugar de partido

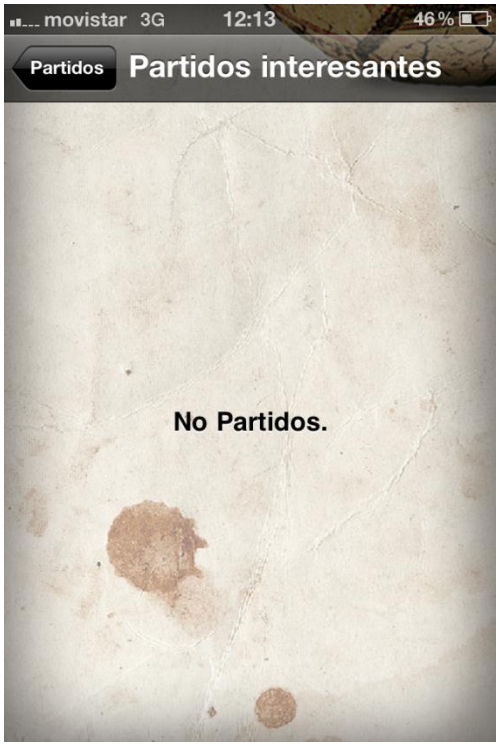


Figura 23. Recomendador de partidos

Sitio web

El sitio web de Fubles es bastante más completo y fiable que el contenido que se ofrece en la aplicación. Prueba de ello es que el número de usuarios es bastante elevado, siendo líder en este aspecto (más de 130.000), y desde su origen ya se han organizado más de 26.000 eventos, lo que les hace ser expertos en la materia.

Enlace al sitio web de Fubles

LovingSports

LovingSports es un sitio web cuya funcionalidad va un paso más allá de lo visto hasta

ahora en esta sección. Además de organizar eventos deportivos y encontrar gente con la que compartir gustos, puedes establecer un entrenamiento adecuado a tu caso o incluso realizar compra-venta de material deportivo. Las interfaces son bastante atractivas e intuitivas, tanto en la versión web, como en la disponible para dispositivos móviles (figuras 24 y 25).

Figura 24. Presentación LovingSports

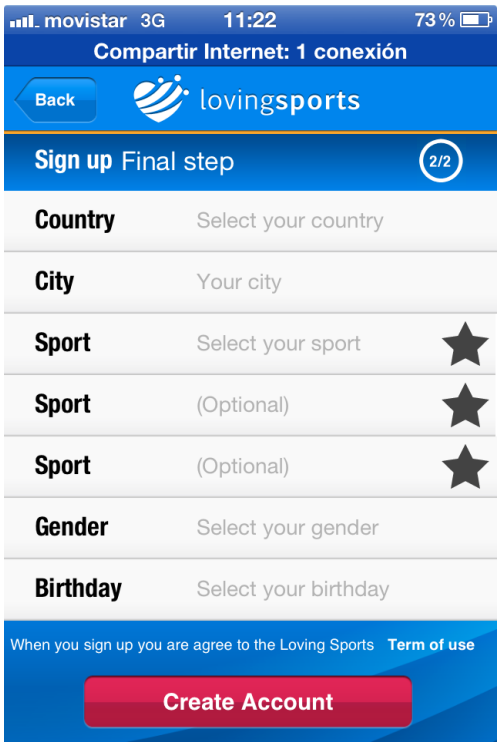


Figura 25. Registro LovingSports

El problema es que en estos momentos no está disponible la aplicación en App Store. Ha sido retirada hace poco. Afortunadamente, nosotros pudimos descargarla a tiempo aunque la evaluación que realizamos no fue buena. No se puede utilizar el mismo usuario y contraseña con el que, acceder a través de la web, y tampoco se puede crear nueva cuenta (figuras 26 y 27). No porque no exista la opción, sino porque se producen todo tipo de fallos e incluso en algunos aspectos tratamos con una aplicación incompleta. Por este motivo no podemos realizar un análisis más exhaustivo. No obstante, dejamos el enlace para quien quiera echarle un vistazo.

Sitio web LovingSports

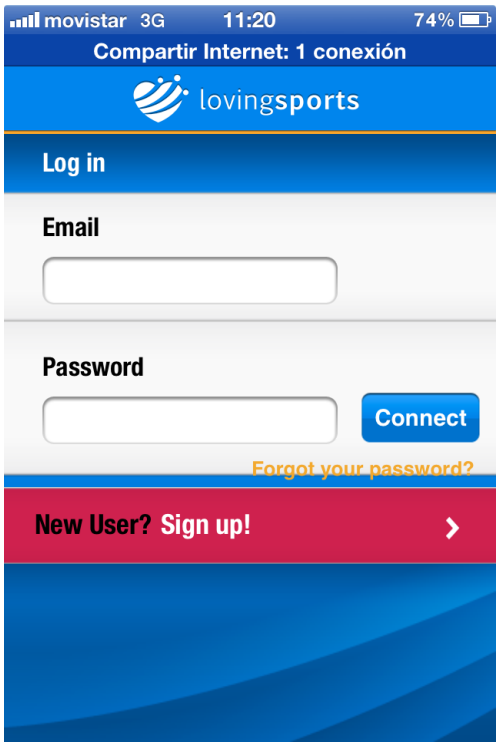


Figura 26. Acceso a LovingSports

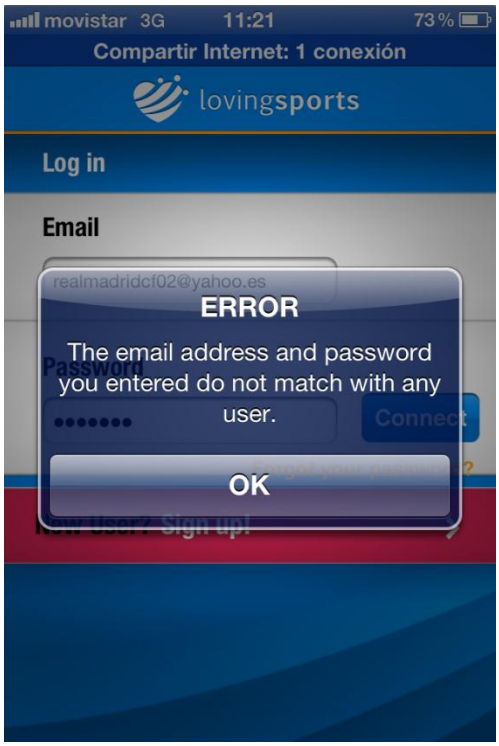


Figura 27. Error de acceso de LovingSports

Conclusiones

La principal conclusión que podemos obtener, tras realizar este análisis sobre aplicaciones disponibles ya en el mercado, es que la competencia es buena, y que los "rivales" son los primeros que te hacen mejorar y querer seguir trabajando. A menudo sirven como referencia, tanto para lo bueno, como para lo malo. El que los usuarios escojan tu producto o no, depende, en gran medida de la competencia.

Por otro lado, estamos ante una aplicación prácticamente nueva en el mercado. Hasta hace bien poco, no se podía encontrar un producto similar disponible para dispositivos móviles. Es por eso que no existe ya un monopolio a cualquier nivel y de cualquier tipo concreto (como sí sucede, por ejemplo, con Facebook o Twitter), lo que permite captar nuevos usuarios con más facilidad, y lo que es más importante, que no prejuzguen sin probar y piensen que una aplicación es mejor que otra sólo por ser más conocida.

LA APLICACIÓN

SocialSport es una red social deportiva para Android que permite gestionar eventos deportivos de forma sencilla e interactuar a usuarios que desean practicar el mismo deporte. También incorpora un sistema de reservas de las instalaciones deportivas Paraninfo Norte y Paraninfo Sur, pertenecientes a la Universidad Complutense de Madrid. Además, posee un sistema de notificaciones que mantiene al usuario actualizado en todo momento de las novedades que se producen.

Herramientas, requisitos y recursos

En este apartado describiremos cuales han sido los recursos software, físicos y humanos empleados en el desarrollo de este proyecto. Para cada una de las elecciones que hemos tomado incluimos una justificación.

Software

En el apartado software comenzaremos hablando de los entornos de desarrollo elegidos para cada parte del proyecto. Para la parte de la aplicación desarrollada en JAVA y XML hemos utilizado Eclipse como entorno de desarrollo ya que, además de estar familiarizados con él y ser gratuito, nos permite instalar fácilmente el plugin (o herramienta añadida al entorno de desarrollo) *Android Development Toolkit* (ADT), diseñado expresamente para desarrollar y depurar aplicaciones e interfaces gráficas para Android en Eclipse. Este plugin está disponible en el sitio web:

http://developer.android.com/sdk/eclipse-adt.html

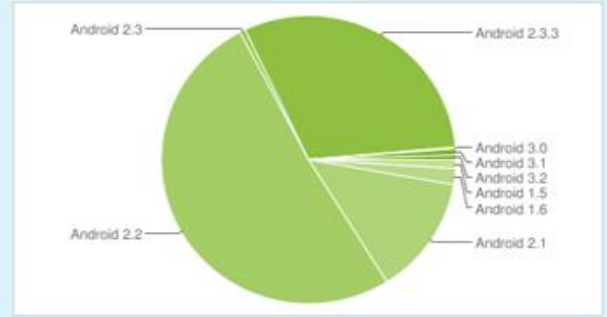
Una vez elegido el entorno de desarrollo para la parte de la aplicación, ha sido necesario descargar el paquete *Android Software Development Kit* (SDK) que contiene las librerías necesarias para desarrollar aplicaciones para Android junto con el *Android Virtual Device* (AVD), un simulador de dispositivos virtuales Android. La versión elegida del Android SDK fue en un principio la *API level 8* que contiene las librerías estándar para la versión 2.2 de Android, la más distribuida al comienzo del desarrollo de nuestro proyecto (Figura 28), pero tras decidir incluir localización a través de *Google Maps* en nuestra aplicación pasamos a usar el paquete *Google APIs* también para la versión 2.2. Ambos paquetes pueden descargarse desde el sitio web:

http://developer.android.com/sdk/index.html

Otra herramienta software que hemos necesitado en el desarrollo de la parte Java de la aplicación ha sido la librería *GreenDroid* desarrollada por Cyril Mottier, que

implementa una barra de menú en la parte superior de la aplicación para versiones Android anteriores a la 3.0 (*API level 11*), ya que esta barra no se incluyó oficialmente hasta esta versión. Decidimos utilizar esta librería porque queríamos que la experiencia de usuario de nuestra aplicación fuese similar a la de la mayoría de las aplicaciones de Android y que siguiese las pautas de diseño de interfaces gráficas recomendadas por Google. Esta librería, además de cumplir esto, es una de las más estables, conocidas, utilizadas, personalizables y, probablemente, la que más actualizaciones y mejoras recibe de su creador. Al igual que con la versión del *Android SDK*, comenzamos utilizando el paquete estándar de *GreenDroid* y tras incluir *Google Maps* en la aplicación pasamos a utilizar el paquete *GreenDroid con Google APIs*. La librería puede descargarse desde:

https://github.com/cyrilmottier/GreenDroid



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	1.0%
Android 1.6	Donut	4	1.8%
Android 2.1	Eclair	7	13.3%
Android 2.2	Froyo	8	51.2%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.6%
Android 2.3.3 - Android 2.3.4		10	30.7%
Android 3.0	Honeycomb	11	0.2%
Android 3.1		12	0.7%
Android 3.2		13	0.5%

Data collected during a 14-day period ending on September 2, 2011

Figura 28. Gráfica representativa de distribución de versiones del sistema operativo Android

En el desarrollo de las interfaces gráficas se ha empleado el plugin *Android Development Toolkit* (ADT) integrado en Eclipse, como ya se ha comentado anteriormente, que nos permite crear y editar los archivos XML en los que están codificadas estas interfaces. Este plugin también nos permite manipular estas mismas interfaces de forma gráfica incluyendo y moviendo los distintos elementos con el ratón. A parte del ADT, se ha necesitado usar el programa de edición gráfica *Adobe Photoshop* para crear algunas de las imágenes empleadas en la interfaz gráfica de usuario.

Para la parte de la aplicación alojada en el servidor, se ha utilizado un entorno de desarrollo para código PHP y con Eclipse, ya que es el usado también para la parte en código JAVA. Además, para poder subir este código al servidor, elegimos el cliente FTP

FileZilla ya que es gratuito y muy utilizado. Este cliente FTP puede descargarse del sitio web:

http://filezilla-project.org

La base de datos que contiene la información de la red social se ha desarrollado en MySQL ya que es un entorno gratuito en el que teníamos experiencia previa. Para manipular la base de datos hemos utilizado el entorno de desarrollo *MySQL Workbench* que es oficial, gratuito y permite un nivel de administración de la base de datos total. Esta herramienta puede descargarse desde el sitio web:

http://mysqlworkbench.org

Para alojar tanto la base de datos como la funcionalidad codificada en PHP hemos escogido un servidor de pago para ganar en fiabilidad y privacidad de los datos. El servidor escogido está montado sobre sistema operativo *Linux*, ofrece soporte para bases de datos *MySQL* y posee un servidor web *Apache* para nuestro código PHP. El sitio web de la empresa de hosting que nos proporciona el servicio es:

http://www.loading.es

Otro de los recursos empleados en el desarrollo del proyecto ha sido un sistema de control de versiones o repositorio con el fin de compartir el código entre los miembros del equipo. El sistema elegido ha sido *Subversion* de Apache con el que ya habíamos trabajado anteriormente, y el repositorio escogido es de la compañía Assembla, debido a que ofrece un servicio gratuito pero con privacidad para el código. Como cliente de este repositorio hemos utilizado el plugin *Subclipse* para Eclipse ya que, una vez más, es gratuito, ampliamente usado y poseíamos experiencia en él. El sitio web de Assembla es:

https://www.assembla.com

Y el del el plugin para eclipse es:

http://subclipse.tigris.org

Hardware

Los recursos físicos empleados han sido, además de ordenadores con conexión a internet, dos dispositivos móviles Android que poseíamos con el fin de desarrollar pruebas más allá del dispositivo virtual AVD incluido en el *Android Software Development Kit* (SDK). Hemos utilizado un teléfono móvil *LG Optimus 2X* con las versiones 2.2 y 2.3 de Android y una tablet *Samsung Galaxy Tab 10.1* con la versión 3.1

(Figura 29 y Figura 30).

Figura 29. LG Optimus 2X

Figura 30. Samsung Galaxy Tab 10.1

Humanos

Por último, los recursos humanos empleados hemos sido nosotros mismos: tres desarrolladores con conocimientos en lenguajes Java, PHP, SQL, XML, en bases de datos MySQL y, tras el proyecto, también en desarrollo de aplicaciones Android. En el desarrollo de este proyecto cada uno de los miembros del equipo ha empleado unas 400 horas. Uno de los miembros dedicó, aproximadamente, tres cuartas partes del tiempo empleado en tareas de servidor y base de datos, y una cuarta parte al desarrollo de la aplicación en Android. Sin embargo, los otros dos miembros dedicaron más tiempo al desarrollo de la aplicación en Android, ya que requiere más dedicación.

Características

Facilidades. En todo momento y lugar, puedes buscar compañeros para un partido, encontrar eventos, o sencillamente crearlos. Sólo con disponer de cualquier conexión a Internet en el dispositivo. Y todo de manera muy rápida, sin perder tiempo innecesariamente.

Información en tiempo real. Con el sistema de avisos, las novedades surgidas en torno a tu perfil de usuario serán notificadas en tu dispositivo. Nuevos partidos, nuevos mensajes, etcétera. Además, al acceder a la aplicación, un icono característico, de color rojo y circular, marcarán aquellos lugares en los que se haya producido algún

cambio.

Figura 31. Notificación PUSH SocialSport

Perfil social. Las funcionalidades más básicas de las redes sociales seguirán presentes, incluyendo diferentes vías de comunicación escrita. Además, se pueden seleccionar los deportes a los que se es afín, e incluso realizar una estimación de nivel. También están presentes opciones más comunes como obtener la ubicación, cambiar la imagen de perfil, etcétera.

Perfil grupal. Un usuario puede formar parte de algún grupo (ya sea un equipo o de otro tipo), y estar en contacto permanente con el resto de miembros. Esto también está disponible para los jugadores que se han incorporado a un encuentro, por lo que supone una forma más de permanecer informado.

Desarrollo constante. Desde los inicios de la aplicación, el desarrollo de la misma y la incorporación de nuevas funcionalidades ha sido incesante y va a seguir siéndolo. La constancia en la búsqueda de mejoras acaba dando sus frutos.

Reservas. La aplicación incluye una simulación de reservas sobre 2 de las instalaciones de la Universidad Complutense de Madrid, teniendo en cuenta horarios de apertura, deportes disponibles, pistas existentes… todo acorde con la realidad. Esto sienta una base para el futuro que puede resultar beneficiosa para la UCM, si acaba explotándose.

Acompañantes. Esta funcionalidad a priori tan simple puede llegar a ser muy útil a la hora de encontrar participantes en un evento, puesto que la aplicación no exige que los acompañantes estén registrados. Un usuario puede incorporarse a un evento deportivo e indicar que llevará compañía, sin necesidad de dar más explicaciones, y será tenido en cuenta en todo momento.

Otras características a destacar son la posibilidad de rotar la pantalla sin que se vea alterado el comportamiento, la incorporación de Google Maps al sistema con el fin de facilitar la obtención de la ubicación, la disponibilidad de la aplicación en inglés, el aspecto fresco y dinámico de las interfaces, sin llegar a ser cargante, o la facilidad de uso.

Figura 32. Recursos y permisos necesarios

Figura 33. Recursos y permisos necesarios

Estructura

Dado que es una aplicación para dispositivos móviles que utiliza datos alojados en un servidor web, hemos dividido el desarrollo de la misma en cuatro ámbitos: funcionalidad en la aplicación y la interfaz gráfica de usuario, funcionalidad en el servidor y base de datos.

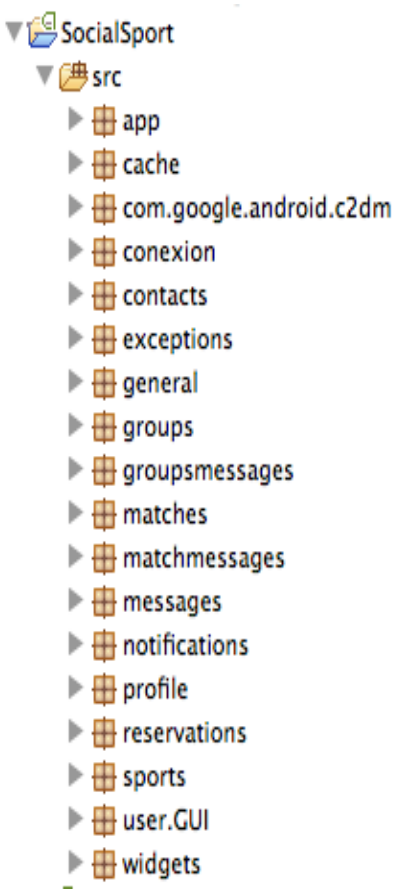


Figura 34. Paquetes Java

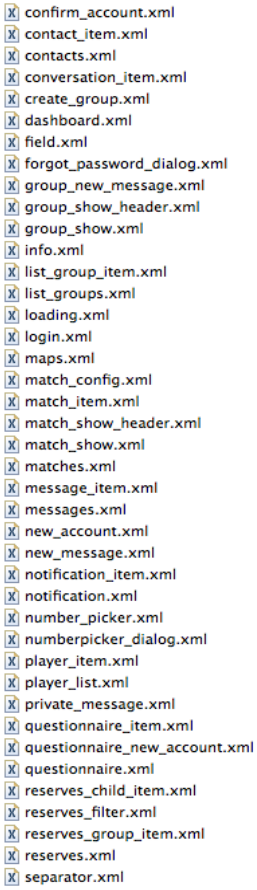


Figura 35. Clases XML

Funcionalidad en la aplicación e interfaz gráfica de usuario

Estos dos ámbitos se han implementado en JAVA. Es la parte de la aplicación que se instala en el dispositivo Android. Se ha utilizado el patrón de diseño "Modelo-Vista-Controlador". El "Modelo" y el "Controlador" del patrón se encuentran en las clases JAVA. Consta de dieciocho paquetes y ciento cinco clases. La "Vista" (la interfaz gráfica de usuario) está compuesta por cuarenta y cinco ficheros "XML".

- **app:** Paquete que contiene la clase **MyApp.java**. Esta clase contiene en todo momento el contexto de la aplicación.
- **cache:** Paquete que contiene las clases que implementan una estructura de

datos que almacena elementos cuyo uso es frecuente.

- **Images.java**: Estructura que contiene las imágenes de usuario y de grupos que más consulta el usuario del dispositivo.
- **ImagesCache.java:** Clase que gestiona la caché de imágenes. Hay un máximo número de imágenes que se pueden almacenar. Cuando ya se ha llegado a ese número, se elimina aquella imagen que más tiempo lleva sin ser referenciada (LRU).
- **PairImageTime.java:** Clase que contiene una imagen y el instante que ha sido referenciada dicha imagen por última vez.

- **com.google.android.c2dm:** Este paquete contiene las clases necesarias para implementar las notificaciones PUSH en Android. Éstas se usan para avisar a los dispositivos que se ha producido un cambio en el servidor que afecta a los usuarios de los mismos. De esta manera, el dispositivo puede notificar al miembro dichos cambios sin necesidad de realizar consultas al servidor cada cierto tiempo, ahorrando así energía.

- **conexión:** Paquete que contiene las clases necesarias para realizar una consulta a un servidor web.
  - **Conexion.java:** Clase que gestiona la conexión de la aplicación con servidores web. Se usa para obtener la información que se mostrará al usuario. Cuando es necesario, traduce a una estructura de datos procesable por el resto de clases de la aplicación, la información recibida del servidor.
  - **MyNameValuePair.java:** Clase que implementa la interfaz NameValuePair. Se usa para enviar datos al servidor web (en este caso, se utiliza el método POST).

- **contacts**: Las clases de este paquete implementan la funcionalidad del módulo "Usuarios".
  - **ContactItem.java:** Clase que almacena la información de un usuario concreto que se mostrará en la lista de contactos.
  - **Contacts.java:** Esta clase gestiona la lista de los contactos. Carga dicha lista con la información necesaria y controla los eventos de la misma. Es el "Controlador" en el patrón de diseño anteriormente descrito del módulo de "Usuarios".
  - **TabContacts.java:** Es la clase que contiene las pestañas "Mis amigos", "Todos los usuarios" y "Solicitudes". Gestiona el funcionamiento de las mismas.
  - **Users.java:** Es el "Modelo" del módulo de "Usuarios". Gestiona las consultas al servidor y el tratamiento de los datos en éste.

- **exceptions:** Paquete que alberga las excepciones propias de la aplicación.
  - **OffLineException.java:** Hereda de la clase Exception. Esta excepción se lanza cuando la aplicación ha detectado que el dispositivo no dispone de conexión a Internet.

- **general:** Este paquete contiene las clases de las que heredan otras clases de distintos paquetes.
  - **Consults.java:** De esta clase heredan todas aquellas clases que hacen de "Modelo", es decir, aquellas que consultan los datos al servidor y los tratan para su posterior interacción con el usuario.
  - **ListITem.java:** Interfaz que implementan todas aquellas clases que son elementos de una lista.
  - **NotificationItem.java:** Implementa ListItem. Esta clase muestra, cuando es necesario, notificaciones en las listas.
  - **SeparatorItem.java:** Es el separador en las listas. Por ejemplo, en la lista que aparece en el perfil de usuario, hay tres separadores: "Deportes", "Partidos" y "Grupos".

- **groups:** En este paquete se encuentran las clases que implementan la funcionalidad de los grupos.
  - **GroupItem.java:** Contiene la información de un grupo. Esta clase se utiliza para mostrar los grupos como elementos de una lista.
  - **Groups.java:** Es el "Modelo" del módulo "Grupos". Gestiona la petición al servidor de datos y el tratamiento de los mismos.
  - **ListGroups.java:** Es el "Controlador" de la lista de grupos. Se encarga de proporcionar la información a la lista y de gestionar los eventos de la misma.
  - **NewGroup.java:** Es el "Controlador" de la creación de un nuevo grupo.
  - **ShowGroup.java:** Es el "Controlador" de la parte de la aplicación en la que el usuario interactúa con los grupos. Puede escribir un comentario, consultar los miembros del grupo o invitar a otros usuarios.
  - **TabGroups.java:** Clase que gestiona las pestañas de los grupos: "Mis grupos" e "Invitaciones".

- **groupsmessages:** Paquete que implementa los comentarios en los grupos.

- **matches:** Paquete que implementa el "Modelo" y el "Controlador" del módulo "Partidos".
  - **MatchConfig.java**: Esta clase implementa el "Controlador" de la creación y edición de partidos.
  - **Matches.java:** Implementa el "Modelo" del módulo "Partidos". Realiza la consulta de información al servidor y el tratamiento de ésta para que pueda ser utilizada por los "Controladores" que lo soliciten.
  - **MatchesList.java:** Es el "Controlador" de la lista de partidos. Solicita información al "Modelo" y se la proporciona a la "Vista". Gestiona los eventos de la lista.
  - **MatchShow.java:** En esta clase se muestra la información de un partido. Los usuarios pueden escribir comentarios, unirse al partido, añadir acompañantes y consultar los participantes.
  - **PlayerItem.java:** Clase que contiene la información de un participante del partido. Esta información se mostrará en una lista.
  - **PlayerList.java:** "Controlador" de la lista que muestra los participantes

de un partido.
    - **TabMatches.java:** Clase que gestiona las pestañas de los partidos.

- **matchmessages:** Paquete que implementa los comentarios de los partidos.

- **messages:** Paquete que implementa el "Modelo" y el "Controlador" del módulo "Mensajes privados".
    - **ConversationItem.java:** Clase que almacena la información de una conversación entre dos usuarios. Esta información se mostrará en una lista en la que aparecerán las últimas quince conversaciones privadas que ha tenido un usuario.
    - **Conversations.java:** Implementa el "Controlador" de la lista de conversaciones privadas de un usuario. Proporciona la información a la lista y gestiona sus eventos.
    - **MessageItem.java:** Alberga la información de un mensaje privado dentro de una conversación.
    - **Messages.java:** Implementa el "Modelo" del módulo "Mensajes privados". Realiza las consultas de información al servidor y trata esta información para proporcionársela a los "Controladores" que lo soliciten.
    - **NewMessage.java:** Clase que implementa la funcionalidad para el envío de un nuevo mensaje privado a un usuario.
    - **PrivateMessages.java:** Implementa el "Controlador" de la lista de mensajes privados dentro de una conversación.

- **notifications:** Paquete que implementa las notificaciones de la aplicación. Se usan para notificar al usuario su novedades (mensajes privados nuevos, peticiones de amistad, invitaciones a partidos y a grupos, mensajes nuevos en partidos y grupos, etc.).
    - **AllNotifications.java:** Clase que implementa el patrón Singleton. Se utiliza con el fin de almacenar las notificaciones que tiene el usuario para poderlas mostrar en diferentes vistas de la aplicación sin necesidad de tener que consultarlas al servidor cada vez que una vista las solicite. De esta forma, solo se realiza la conexión con el servidor cuando es estrictamente necesario.
    - **Notifications.java:** Guarda la información necesaria para mostrar las notificaciones.
    - **NotificationsConsult.java:** Clase que implementa en "Modelo" de las notificaciones. Realiza la consulta al servidor y el tratamiento de la información que obtiene de éste. A continuación la almacena en la clase AllNotifications.
    - **NotificationsTypes.java:** Clase que contiene las constantes utilizadas para diferenciar los distintos tipos de notificaciones.

- **profile:** Implementa la funcionalidad del módulo "Perfil de usuario".
    - **Locations.java:** Clase utilizada para almacenar la información de una localización. Guarda las coordenadas, la dirección, la ciudad y el país.

- **Maps.java:** Implementa el uso de Google Maps en la aplicación. Es la clase donde se utiliza el API de Google. Se da la opción al usuario de introducir manualmente su ubicación utilizando los mapas en lugar de obtenerla utilizando el GPS u otros métodos del dispositivo.
- **MyProfile.java:** Utiliza el patrón Singleton. Hereda de la clase Profile. Guarda los datos del usuario del dispositivo. Esta información se encuentra disponible en todo momento en cualquier clase de la aplicación. De esta manera, no es necesario realizar una consulta al servidor para obtener dicha información cada vez que se quiera obtener un dato del usuario.
- **Profile.java:** Clase que realiza la consulta de los datos de un perfil y los almacena.
- **Questionnaire.java:** Implementa el "Controlador" de la elección de deportes y nivel de los mismos.
- **Settings.java:** Esta clase implementa la edición de los datos del perfil de usuario.
- **UserProfile.java:** Implementa el "Controlador" de la visualización del perfil de usuario. Muestra la imagen, el nombre y la última ubicación del usuario junto con sus intereses.

- **reservations:** Paquete que implementa las reservas de pistas.
  - **Field.java:** Implementa el "Controlador" de la vista que muestra la información de un campo. Muestra la organización a la que pertenece, el nombre del campo, la hora que se ha elegido para la reserva, la dirección y las tarifas disponibles.
  - **RateItem.java:** Clase que almacena la información de una tarifa determinada.
  - **Reservations.java:** Clase que implementa el "Modelo" del módulo "Reservas". Realiza las consultas al servidor para extraer la información necesaria. Implementa los algoritmos utilizados para la ordenación de los campos.
  - **ReservationsList.java:** Implementa el "Controlador" de la lista que muestra los campos disponibles en función de los parámetros elegidos por el usuario.
  - **ReserveFilter.java:** Es la clase utilizada para realizar un filtrado de los campos disponibles. Se pide al usuario que elija un deporte, una fecha y un intervalo de horas para poder realizar la consulta.
  - **ReserveItem.java:** Clase que almacena la información relativa a una reserva.

- **sports:** Paquete que implementa la funcionalidad de los deportes.
  - **AllSports.java:** Clase implementada con el patrón Singleton. Contiene los datos de los deportes disponibles. Realiza una consulta al servidor al iniciar la aplicación y mantiene los datos disponibles para cualquier clase que lo necesite.
  - **LevelSport.java:** Contiene el nivel de un deporte determinado para un usuario determinado.

- **SportItem.java:** Contiene el nombre y el identificador de un deporte.
- **SportsConsult.java:** Clase que realiza las consultas al servidor y trata la información proporcionada por éste. Consulta los deportes disponibles en la base de datos, actualiza los niveles de los deportes de un usuario, consulta los deportes favoritos de un usuario, etc.

- **user.GUI:** Contiene las clases principales de la aplicación.
  - **C2DMReceiver.java:** Esta clase genera las notificaciones que avisan al usuario de cambios en su cuenta que le pueden interesar. Cuando detecta que el dispositivo ha recibido una notificación PUSH enviada a la aplicación SocialSport, recoge la información necesaria y se la muestra al usuario.
  - **ConfirmAccount.java:** Clase que muestra un mensaje al usuario indicando que la creación de la nueva cuenta se ha realizado correctamente.
  - **ConnectedActivity.java:** Clase de la que heredan todos los "Controladores" anteriormente descritos. Contiene métodos utilizados por diversas clases.
  - **ConnectedMapActiviy.java:** Clase de la que heredan aquellas clases que contienen mapas.
  - **ConnectedTabActivity.java:** Clase de la que heredan aquellas clases que gestionan pestañas.
  - **Dashboard.java:** Es el "Controlador" de la "Vista" del menú. Gestiona los eventos y actualiza las notificaciones.
  - **Login.java:** Clase que gestiona la autenticación de un usuario. Utiliza los datos introducidos por el mismo y, mediante una conexión con el servidor, determina si esos datos son correctos para poder acceder a la información.
  - **Newccount.java:** Clase que gestiona la creación de una nueva cuenta. Recoge los datos introducidos por el usuario a través de la interfaz gráfica y realiza la conexión necesaria con el servidor para crear un nuevo usuario.
  - **SplashScreen.java:** Vista inicial de la aplicación en la que aparece el nombre de la misma.

- **widgets:** Este paquete contiene clases que implementan elementos de la interfaz gráfica modificados por los propios desarrolladores de la aplicación.

Funcionalidad en el servidor

Este ámbito es el responsable de la gestión de la información almacenada en la base de datos. Funciona de enlace entre ésta y la aplicación. Recoge la información enviada por la aplicación, la procesa, realiza la consulta a la base de datos y, cuando es necesario, recoge información y la trata para poder proporcionársela a la aplicación. Se ha desarrollado en PHP. Está compuesto por diecisiete paquetes y cuarenta y seis

ficheros.

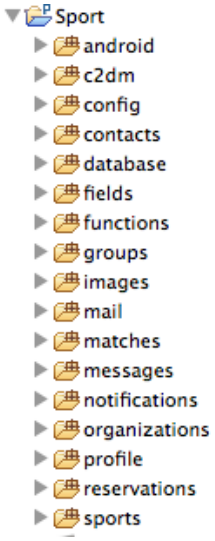


Figura 36. Paquetes php

- **android:** Paquete que contiene las interfaces con las que la aplicación se comunica con el servidor. Éstas recogen la información enviada por la aplicación e invocan las funciones que correspondan. Si es necesario, tratan los datos devueltos por estas funciones para enviárselos a la aplicación y que ésta los pueda procesar.
  - **addfriendandroid.php:** Interfaz utilizada para enviar solicitudes de amistad.
  - **consultswithoutpassword.php:** Se utiliza por la aplicación cuando se necesita comprobar algún dato antes de haber realizado la autenticación.
  - **friendsandroid.php:** Gestiona todo lo que tiene que ver con las relaciones de amistad entre los usuarios.
  - **getprivatemessagesandroid.php: I**nterfaz utilizada para consultar las conversaciones privadas entre usuarios.
  - **getsportsandroid.php:** Se utiliza para consultar los deportes disponibles cuando el usuario aún no se ha registrado (es utilizada en la creación de una nueva cuenta cuando el usuario tiene que seleccionar sus deportes favoritos).
  - **groupsandroid.php:** Interfaz que gestiona la funcionalidad de los grupos.
  - **loginandroid.php:** Se utiliza para consultar la validez de los datos introducidos por el usuario en la autenticación.
  - **matchesandroid.php:** Interfaz dedicada a la gestión de los partidos.
  - **newprofileandroid.php:** Interfaz utilizada para crear una nueva cuenta.
  - **notificationsandroid.php:** se utiliza para consultar las notificaciones de un usuario.
  - **profileandroid.php:** Interfaz que gestiona los perfiles.

  - **recoverpasswordandroid.php:** Se utiliza para la recuperación de contraseñas.
  - **registrationpshandroid.php:** Interfaz que registra dispositivos asociados a un usuario para poder enviarle notificaciones PUSH.
  - **reservationsandroid.php:** Gestiona las reservas.
  - **sendprivatemessageandroid.php:** Interfaz utilizada para el envío de mensajes privados entre usuarios.
  - **sportsadnroid.php:** Interfaz que gestiona los deportes.

- **c2dm:** Paquete que implementa las notificaciones PUSH en la parte del servidor.

- **config:** Paquete que contiene un fichero de configuración.

- **contacts:** Los ficheros que alberga implementan la funcionalidad de las relaciones de amistad.
  - **findfriends.php:** Se utiliza para realizar búsquedas entre los usuarios.

- **database:** Paquete que implementa la conexión con la base de datos.
  - **connector.php:** Contiene la clase utilizada para establecer una conexión con la base de datos.
  - **consult.php:** Implementa funciones que utilizan la clase anteriormente descrita para realizar consultas a la base de datos.

- **fields:** Sus ficheros implementan la funcionalidad de los campos.

- **functions:** Contiene funciones utilizadas por otro ficheros.

- **groups:** Paquete que implementa la funcionalidad de los grupos.
  - **group.php:** Implementa la clase Group. Ésta alberga los datos de un grupo y los modifica si hay necesidad.

- **images:** Directorio donde se almacenan las fotos subidas por los usuarios.

- **mail:** Paquete utilizado para enviar el e-mail de confirmación de creación de cuenta y el de recuperación de contraseña.

- **matches:** Paquete que gestiona la funcionalidad de los partidos.
  - **matches.php:** Implementa la clase Match. Ésta almacena y modifica los datos de un partido determinado. También proporciona toda la información que se le solicite sobre el mismo.

- **messages:** Implementa la funcionalidad de los mensajes privados entre usuarios.

- **notifications:** Paquete que implementa las funciones que proporcionan la

información requerida sobre las notificaciones.

- **organizations:** Implementa la clase que gestiona la información de una organización.

- **profile:** Paquete que implementa toda la funcionalidad relacionada con la creación, edición y visualización de perfiles.
  - **login.php:** Implementa la autenticación de un usuario. Coteja la información que se recibe con la existente en la base de datos.
  - **profile.php:** Implementa la clase Profile, que gestiona los datos de un perfil.

- **reservations:** Contiene la funcionalidad de las reservas.
  - **ressocialsport.php:** Implementa la gestión de las reservas de SocialSport, es decir, gestiona los campos y horarios registrados en la base de datos de SocialSport (en esta versión solo existe una simulación de las instalaciones del Paraninfo Norte y el Paraninfo Sur de la Universidad Complutense de Madrid). Contiene los algoritmos necesarios para simular los cuadrantes utilizados en la realidad para la gestión de las reservas.
- **sports:** Paquete que contiene los ficheros que implementan la funcionalidad relacionada con los deportes.

Base de datos

La base de datos almacena toda la información necesaria en los otros ámbitos. Es el núcleo de la herramienta. Hemos implementado una base de datos relacional, utilizando el lenguaje MySQL. Está compuesta por veintisiete tablas.

- **sports:** Almacena el nombre de los deportes.

- **profile:** Almacena los datos de un perfil.

- **friend:** Es utilizada para establecer una relación de amistad entre dos usuarios.

- **privateconversation:** Almacena las conversaciones entre dos usuarios.

- **groups:** Almacena la información de los grupos.

- **membergroup:** Almacena los perfiles que son miembros de un grupo.

- **groupinvitations:** Es utilizada para las invitaciones a grupos.

- **publicationgroup:** Contiene los comentarios en los grupos.

- **matches:** Almacena la información de los partidos.
- **membermatch:** Almacena los perfiles que participan en un partido.
- **matchinvitations:** Se utiliza para las invitaciones a partidos.
- **matchmessages:** Almacena los comentarios en los partidos.
- **fields:** Almacena la información de los campos que se pueden reservar.
- **reservation:** Almacena las reservas realizadas.
- **push:** Guarda la información que se necesita para enviar notificaciones PUSH.
- **locations:** Guarda las localizaciones de los usuarios.
- **sportslevel:** Almacena los niveles de los deportes favoritos de los usuarios.

Desarrollo

Cronológicamente el desarrollo se comenzó implementando la creación de cuentas y la autenticación de usuarios. En este punto, se tomaron algunas decisiones como utilizar el correo electrónico como nombre de registro, en lugar de un "nick", porque es más sencillo de recordar. Para evitar que se introdujeran cuentas falsas, optamos por enviar un mensaje a la dirección de correo introducida, con un enlace de confirmación al que el usuario debe dirigirse para poder activar su nueva cuenta. En el proceso de creación de un nuevo miembro, se ofrece la opción de seleccionar los deportes favoritos y puntuar el nivel en los mismos, para ofrecer una experiencia personalizada en el uso de la herramienta.

El siguiente módulo es el perfil de usuario junto con sus ajustes. En la visualización del mismo, se ha optado por desmarcarse del típico perfil de usuario de redes sociales en los que se muestra demasiada información. Por este motivo, hemos decidido separar la visualización, de la edición. En el perfil se muestra la imagen, el nombre, la última ciudad que ha elegido el usuario, sus deportes favoritos y los niveles que tiene en los mismo, la media de los niveles de los deportes listados, los partidos que tiene pendientes por jugar y los grupos a los que pertenece. En los ajustes se puede modificar la imagen, el nombre, la contraseña, la localización, los deportes favoritos y los niveles correspondientes, puesto que pueden variar dependiendo de las circunstancias temporales de cada uno.

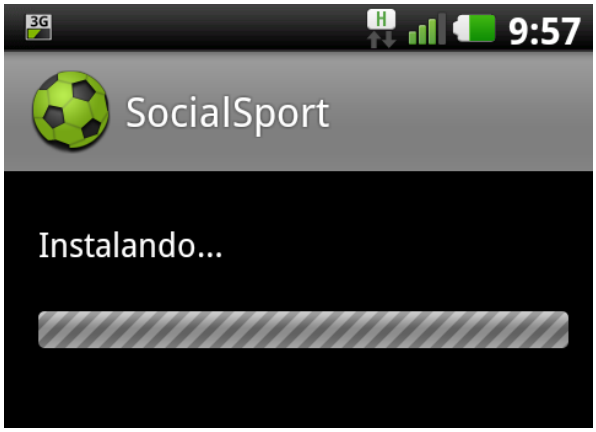


Figura 37. Instalación de la aplicación

Los miembros de la red pueden estar relacionados con otros. Para ello, se ha implementado el módulo de usuarios. Éste consta de tres pestañas: “Amigos”, “Todos los usuarios” y “Solicitudes”. Como bien indican sus títulos, en la primera se encuentran los amigos del usuario, en la segunda se puede buscar a aquellas personas que no tienen relación con el mismo y en la tercera se muestran las peticiones de amistad. Para establecer una relación entre dos usuarios, uno de ellos debe enviar una petición de amistad al otro, y éste debe confirmarla. Estas relaciones se utilizan para invitaciones a partidos y grupos.

En toda red social es necesario que los usuarios puedan comunicarse entre sí. Por ello, hemos decidido implementar un módulo de mensajes privados. De esta forma un usuario puede enviar mensajes privados a todos sus amigos.

El siguiente módulo que hemos desarrollado fue el de los partidos. Consta de tres pestañas: “Mis partidos”, “Públicos” e “Invitaciones”. En la primera se encuentran los partidos que el usuario tiene pendientes para jugar. En la segunda se muestran los partidos públicos de los deportes que el usuario haya seleccionado como favoritos. En la tercera aparecen las invitaciones a partidos. Al crear el partido se puede seleccionar la opción de hacerlo público para que cualquier usuario de la red pueda unirse, si no, sólo se pueden unir aquellos usuarios que hayan sido invitados previamente. Cada partido tiene un número limitado de plazas. Hemos decidido que los participantes del evento pudieran añadir acompañantes, siempre y cuando hubiera plazas libres en el partido. Realizamos esta implementación para que constase la participación en el partido de personas que no tienen acceso a la herramienta.

En ocasiones, es de mucha utilidad congregar a varias personas en un mismo grupo para compartir información. Éste fue el motivo por el que implementamos los grupos.

Se concibió como manera de unir a varios usuarios (ya sea en un equipo o simplemente un grupo social). El usuario que crea el grupo puede dar permisos al mismo para que cualquier miembro pueda invitar a otros usuarios o solo lo puedan hacer los administradores. Para acceder a un grupo, un usuario debe haber recibido una invitación previamente.

Figura 38. Icono de aplicación

Una de las funcionalidades más atractivas es la de notificar al usuario en el momento que se produce una novedad en torno a su perfil. Estar actualizado y ser informado instantáneamente hace mucho más interesante la aplicación, por el simple hecho de que en innumerables ocasiones, alguna persona se pierde eventos interesantes solamente por no ser/estar informado a tiempo. Esta posibilidad nos la ofrece las notificaciones PUSH, que aparecen en la barra de notificaciones de Android, siempre que se produce un cambio en torno a un usuario, en un determinado campo, a nivel de servidor.

Por último, implementamos el módulo de las reservas. La idea inicial fue simular las instalaciones deportivas del Paraninfo Norte y el Paraninfo Sur de la Universidad Complutense de Madrid. Sin embargo, quisimos diseñar este módulo lo más genérico posible para poder añadir en un futuro instalaciones de otros recintos deportivos, con sus ubicaciones y diferentes horarios. Tomamos como ejemplo los cuadrantes que se utilizan actualmente para las reservas de pistas (Figura 39), proporcionado por el servicio de instalaciones deportivas de la UCM.

Cuando se quiere realizar una reserva, el usuario selecciona el deporte que quiere practicar, el intervalo del día que le interesa y la fecha. Mediante un algoritmo implementado en el servidor, se simula el cuadrante mostrado anteriormente para el deporte seleccionado y se comprueba qué campos hay disponibles para la fecha y los intervalos de tiempo elegidos.

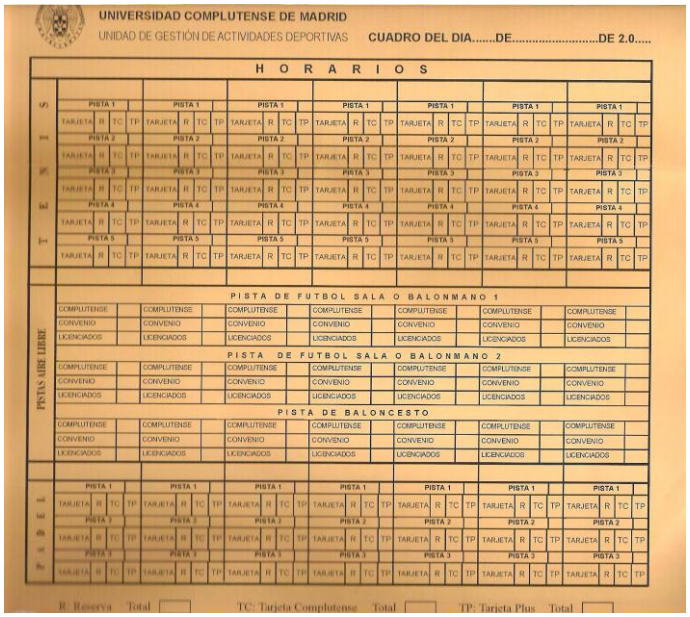


Figura 39.Cuadrante de reservas Paraninfo Norte UCM

Problemas encontrados y soluciones

A lo largo de todo el desarrollo de la aplicación, han ido surgiendo problemas más o menos difíciles de resolver, que han sido solucionados de una manera u otra. Vamos a exponer en esta sección los más destacables.

El primer problema destacable ha sido trabajar con el plugin de Android para Eclipse, denominado Android Development Tools (ADT). A pesar de que Google no cesa en su trabajo, y cada poco tiempo publican una versión mejorada de su producto, todavía tiene muchos aspectos a mejorar. Son frecuentes problemas relativos a bloqueos del Eclipse, de aparición de errores en el código que realmente no son tales, problemas de inclusión de archivos y librerías. También, las dificultades existentes a la hora de realizar las interfaces en XML, para las que el programa no está bien preparado, sobre todo en una parte gráfica que se supone simplifica la labor, y no sólo no es rápida y no funciona bien, sino que su rango de posibilidades es bastante bajo, y al final casi siempre compensa hacerlo todo mediante código, que no está excluido de bloqueos y otros problemas. En este caso, la única solución posible era seguir trabajando hasta que todo funcionara.

Por otro lado, centrándonos en el desarrollo, al empezar a realizar la interacción entre la aplicación y el servidor, nos dimos cuenta de que no era una cuestión sencilla, puesto que resulta difícil encontrar información al respecto, ya sea en libros o en Internet. Además, se suma el problema de que utilizamos lenguaje PHP como

intérprete, y la depuración del mismo a menudo se hace compleja y pesada.

```
/**
 * Muestra los datos de la reserva.
 */
private void showDataReserve() {
    // Se usa este método para que las intrucciones de acceso a la UI se realicen desde el hilo prinipal
    runOnUiThread(new Runnable() {
        public void run() {
            setActionBarContentView(R.layout.field);
            // Imagen
            ImageView image = (ImageView) findViewById(R.id.changeSportImage);
```

Figura 40. Ejemplo de tratamiento de hilos

Al seguir avanzando en el trabajo, estimamos muy necesario el uso de diferentes hilos, de cara, sobre todo, a no resultar una aplicación incómoda e incluso "fea" para el usuario. En concreto, al realizar una carga del servidor, es mucho más "elegante", mostrar una pantalla que informe de lo que se está realizando generando un nuevo hilo. Al finalizar la ejecución del mismo, se cierra y se muestra la nueva pantalla ya actualizada, sin cambios bruscos. Una situación similar se produce al pasar de una vista a otra sin querer cerrar la inicial, por el motivo que sea.

Plantear la situación descrita en el párrafo anterior parece sencillo, pero no lo es a la hora de plasmarlo en el código. Además, la depuración en estos casos puede resultar liosa y confusa. Al final, con atención y muchas horas dedicadas, uno acaba acostumbrándose y se convierte en alguien más hábil, ducho y capaz ante estas situaciones y cada vez resultan menos complejas.

El siguiente problema surge a la hora de subir imágenes al servidor. Por algún motivo que desconocemos, hacerlo desde una aplicación Android resulta complicado, más que por el código, porque una misma implementación funciona en unos casos sí y en otros no, sin seguir un patrón aparente. Al parecer, este problema es muy común entre aquellos que han tratado este tema.

Una vez subidas al servidor, las imágenes son descargadas por la aplicación para ser mostradas al usuario. Este proceso es demasiado lento si la cobertura de red no es buena. Además, hay que tener en cuenta que las imágenes pueden ocupar mucho espacio y eso puede provocar un gran consumo en una tarifa de datos. La solución ha sido implementar una caché que almacena las imágenes que más consulta el usuario y logra agilizar este proceso sin que suponga un gasto extra.

SocialSport ofrece la posibilidad al usuario de obtener su ubicación actual utilizando Google Maps, con el fin de estar localizado, y de que aquel que acceda al perfil de un usuario pueda conocer cuál es su lugar de entorno. Para facilitar esta labor, hay dos

opciones: cargar la propia aplicación de Google Maps o utilizar los mapas como parte de la nuestra. Optamos por la segunda opción, puesto que se ajustaba más a lo que queríamos, a sabiendas de que la complejidad era mayor.

Los problemas los encontramos a la hora de "incrustar" los mapas en nuestra aplicación. Como hemos comentado antes, el plugin de Eclipse no ayuda demasiado, y menos en este caso. Para poder incluirlos dentro de una pantalla, hay que hacerlo de una forma concreta, utilizando una vista dentro de un layout (para más información, consultar anexo Manual Android). Además, hay que generar una clave denominada API Key, que sirve para que se puedan visualizar los mapas. Sin ella, sólo se verá una zona gris en el lugar en el que deberían cargarse.

```
<view
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/mapview"
    class="com.google.android.maps.MapView"
    android:apiKey="@string/APIKEY"
    android:clickable="true"
    android:layout_weight="1"
/>
```

Figura 41. Carga de la vista Google Maps en un layout

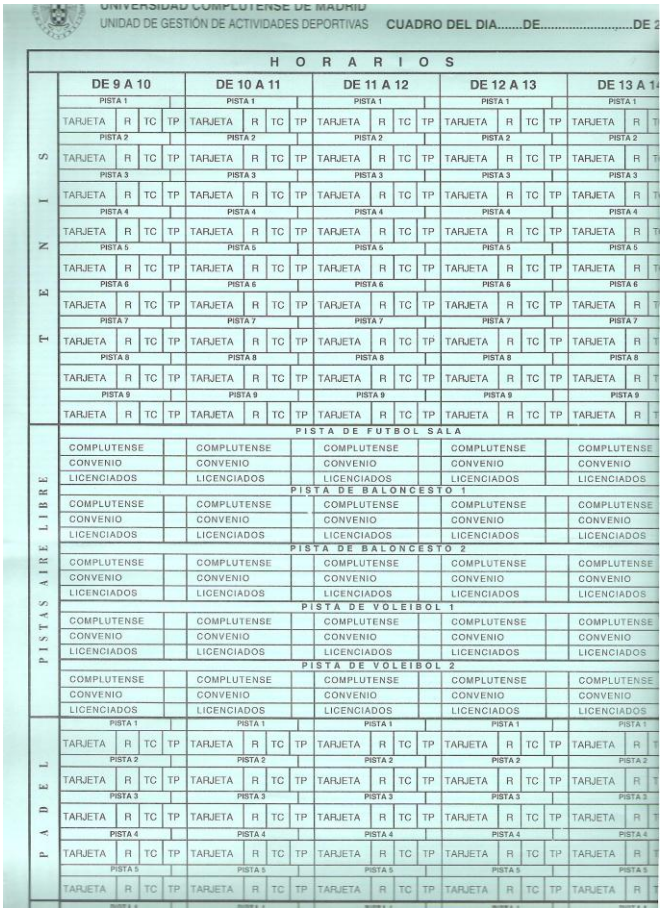
Dando dos vueltas más al manejo de los mapas, decidimos poner un marcador cada vez que el usuario buscara una dirección, con el objetivo de asegurar más fácilmente que el resultado sea satisfactorio. Pero también quisimos, una vez situado un marcador, que con una simple pulsación sobre el mapa, se obtuviera la dirección señalada, de cara a obtener una mayor precisión en la localización. Ambas posibilidades se consiguen creando una clase que herede de una propia y concreta de Google, e interactuando con ella.

También nos planteamos cómo avisar al usuario de las modificaciones en la base de datos que afectaran a su perfil como jugador. En principio, pensamos en mandar SMS o correos electrónicos, pero los primeros supondrían un coste innecesario, y los segundos no son eficientes, puesto que no tienen carácter instantáneo. La solución pasaba por las notificaciones PUSH, que aúnan las características positivas de ambas, el coste cero de los emails, y la instantaneidad de los SMS.

Una vez elegida la opción de las notificaciones PUSH, nos documentamos sobre la implementación de éstas. Google ha desarrollado un sistema que da soporte a este tipo de notificaciones. El primer paso es dar de alta la aplicación en los servidores de Google, mediante un formulario, asociándole una cuenta de correo. A continuación,

desde cada dispositivo se genera un identificador único para dicha aplicación en dicho dispositivo que debe registrarse en los servidores de Google y en el servidor de la aplicación. Cuando se produce un cambio se envía la notificación a los servidores de Google, y éstos la hacen llegar a los dispositivos indicados.

El último problema surgió con las reservas: representar en la base de datos los cuadrantes (Figura 42), teniendo en cuenta que existen pistas en las que se pueden practicar varios deportes (un campo de fútbol sala se puede convertir en dos campos de baloncesto). Como ejemplo, una pista de fútbol sala puede serlo también de baloncesto. Si se contempla la figura, vemos que esta posibilidad no está plasmada en el papel oficial de reservas, pero sí lo está en el algoritmo implementado.


UNIVERSIDAD COMPLUTENSE DE MADRID
UNIDAD DE GESTIÓN DE ACTIVIDADES DEPORTIVAS CUADRO DEL DIA.....DE.....DE 2

H O R A R I O S

	DE 9 A 10	DE 10 A 11	DE 11 A 12	DE 12 A 13	DE 13 A 14
TENIS	PISTA 1 … PISTA 9: TARJETA | R | TC | TP	PISTA 1 … PISTA 9: TARJETA | R | TC | TP	PISTA 1 … PISTA 9: TARJETA | R | TC | TP	PISTA 1 … PISTA 9: TARJETA | R | TC | TP	PISTA 1 … PISTA 9: TARJETA | R | T
PISTAS AIRE LIBRE	PISTA DE FUTBOL SALA	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS
	PISTA DE BALONCESTO 1	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS
	PISTA DE BALONCESTO 2	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS
	PISTA DE VOLEIBOL 1	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS
	PISTA DE VOLEIBOL 2	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS	COMPLUTENSE / CONVENIO / LICENCIADOS
PADEL	PISTA 1 … PISTA 5: TARJETA | R | TC | TP	PISTA 1 … PISTA 5: TARJETA | R | TC | TP	PISTA 1 … PISTA 5: TARJETA | R | TC | TP	PISTA 1 … PISTA 5: TARJETA | R | TC | TP	PISTA 1 … PISTA 5: TARJETA | R | T

Figura 42. Cuadrante de reservas Paraninfo Norte UCM

Por otra parte, al querer hacer el modelo genérico, hay que tener en cuenta que, dependiendo de las instalaciones deportivas y de las pistas que se deseen reservar, los horarios pueden no coincidir y el tiempo de la reserva puede no ser el mismo. Por ejemplo, en unas instalaciones puede que el tiempo de reserva de una pista de tenis

sea de una hora y en otras una hora y media.

Después de muchas propuestas se optó por la siguiente: se crea una organización (puede ser una universidad, el ayuntamiento, un recinto deportivo, etcétera), que establece unas tarifas determinadas para cada deporte. La nueva organización da de alta los campos que quiera incluir en la aplicación, con su localización, sus coordenadas, la primera y la última hora del día a la que se puede alquilar el campo, y la duración de cada reserva. Después, estos campos se dividen en subcampos en los que se especifica el deporte y el nombre del recinto. Se hace esto para poder representar el ejemplo anteriormente descrito: un campo de fútbol sala puede convertirse en dos campos de baloncesto y viceversa. De esta manera se asegura que si alguien realiza una reserva en una pista en la que se puede jugar al fútbol sala o al baloncesto, si se alquila la de un deporte, no se pueda reservar la del otro e igual en caso contrario.

Otros problemas que merece la pena comentar son finalizar la actividad padre, que inicialmente pensamos que no era posible hacerlo, pero después razonamos y descubrimos que las actividades funcionan como los procesos en los sistemas operativos en cuanto a creación y destrucción, por lo que pueden ser finalizados ya sea mediante una señal, o cuando lo haga el hijo; y el otro fue que también trabajamos con el sistema operativo de Apple, y es Eclipse quien elige por defecto la codificación de los caracteres, en función del mismo, por lo que hay que tener cuidado al trabajar con eso. La solución pasaba por declarar todas las cadenas de caracteres en un mismo fichero XML con una codificación concreta.

Estadísticas y datos

Datos del desarrollo

Los datos mostrados a continuación, pueden ser útiles para ser comparados con un futuro desarrollo y permitir realizar una evaluación de calidad del código.

- Líneas de código: 49647
- Líneas de código útiles (sin comentarios ni saltos de línea): 41409
- Porcentaje de líneas útiles: 83,4%
- Clases totales: 151
- Clases JAVA: 105

- Clases PHP: 46
- Paquetes totales: 35
- Paquetes JAVA: 18
- Paquetes PHP: 17
- Tablas base de datos: 27
- Horas totales trabajadas: 1200
- Tiempo laboral total dedicado (en jornadas de 8 horas): 150 jornadas, equivalente a 7 meses aproximadamente
- Fecha inicio desarrollo: 31 de octubre de 2011
- Número de usuarios registrados

Datos de uso

La versión final de esta aplicación se ha distribuido entre personas del ámbito de los desarrolladores. Después de unos días de uso, se realizó una encuesta. Los resultados son los siguientes (siendo 1 muy malo y 5 muy bueno):

- Aspecto visual: 4,5
- Facilidad de uso: 3
- Originalidad: 3,2
- Utilidad: 4,2
- Fiabilidad: 3,8

MANUAL DE USUARIO

Introducción

Esta sección presenta un manual de la aplicación SocialSport, profundizando en todas y cada una de las opciones y funcionalidades que ésta ofrece. El objetivo es facilitar la comprensión de las funcionalidades mediante la visualización de la interfaz.

Acceso

Cuando se ejecuta la aplicación por primera vez, se muestra la pantalla de acceso al sistema (Figura 43). Si el usuario ya está registrado, deberá introducir su email y su contraseña en los campos correspondientemente habilitados para ello, y pulsar el botón "Conectar" para acceder. En caso de no rellenar correctamente los datos, se mostrarán mensajes de aviso (Figura 44).

Figura 43. Acceso a SocialSport

Figura 44. Error en el acceso a la aplicación

En caso contrario, si no se dispone de una cuenta, para darse de alta, es preciso pulsar el botón “Crear cuenta” sin necesidad de rellenar ni el campo “Email”, ni el “Contraseña”, y se producirá una redirección a una nueva pantalla (ver sección “Registro” del manual).

Si el usuario ha olvidado la contraseña, tiene la opción de recuperarla pulsando el texto “Olvidé mi contraseña”. Aparecerá un diálogo en el que debe introducir su dirección de email y pulsar en “OK”. Tras seguir estos pasos, llegará un correo a la cuenta especificada recordando su contraseña.

Nueva cuenta

Para darse de alta como nuevo usuario registrado en la aplicación, es necesario rellenar cada uno los campos mostrados en la Figura 45, y posteriormente pulsar el botón “Registrarse”. Al igual que en el apartado anterior de este manual, en caso de introducir algún campo incorrectamente, se mostrará un aviso por pantalla especificando el error.

Figura 45. Registro de SocialSport

Figura 46. Selección de deportes al registrarse

Posteriormente, aparecerá una pantalla que instará al nuevo registrado a seleccionar los deportes que le interesen, mediante una pulsación en el icono representativo del mismo o sobre su parte derecha, y a autoevaluarse, completando las estrellas existentes para cada elemento (Figura 46). Para completar el registro es preciso pulsar el botón aplicar, que derivará en una pantalla de información.

Si se desea volver a la pantalla correspondiente a la figura 1, basta con presionar el botón “Cancelar”.

Notificaciones

SocialSport posee un sistema propio de notificaciones, de modo que, el usuario pueda estar informado instantáneamente en el momento que se produzcan novedades importantes. Cada vez que recibe un mensaje o una invitación a un partido, el dispositivo suena, vibra o avisa de algún modo de acuerdo a la configuración del equipo que esté utilizando, y quedará plasmado en la barra de estado de su terminal mediante un icono característico (Figura 47).

Figura 47. Recepción de notificación PUSH

Figura 48. Barra de notificaciones desplegada

Desplegando el menú de notificaciones (arrastrando desde la parte superior de la pantalla hacia abajo, Figura 48), aparece la relativa a SocialSport. Pulsando sobre ella, se abrirá la aplicación exactamente en la sección correspondiente al tipo de notificación.

Menú principal

Se puede llegar siempre al menú principal pulsando la tecla "Home" (Figura 49), en la esquina superior derecha de la pantalla. Desde ahí se tiene acceso a todas las funcionalidades de esta aplicación. Los iconos representan, de izquierda a derecha y de arriba abajo, respectivamente: perfil, gente, grupos, campos, reservas, partidos, mensajes, ajustes y desconexión (Figura 50). Pulsando en ellos se accede a cada sección referida, a continuación, en este manual.

Figura 49. Botón "Home"

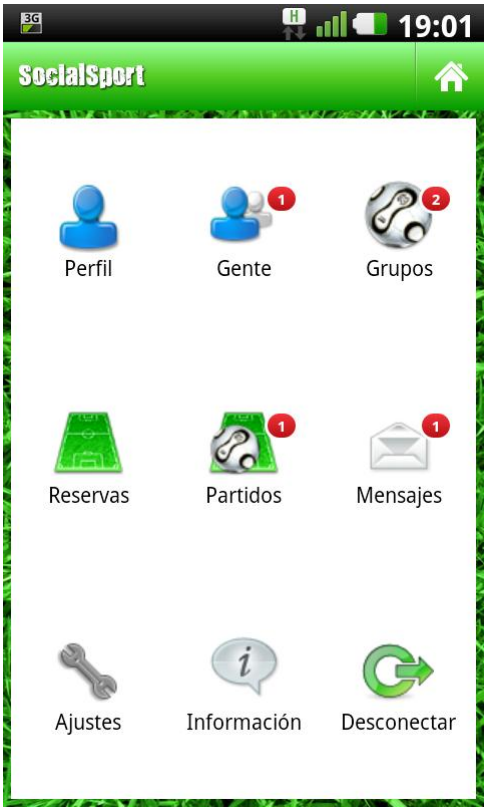


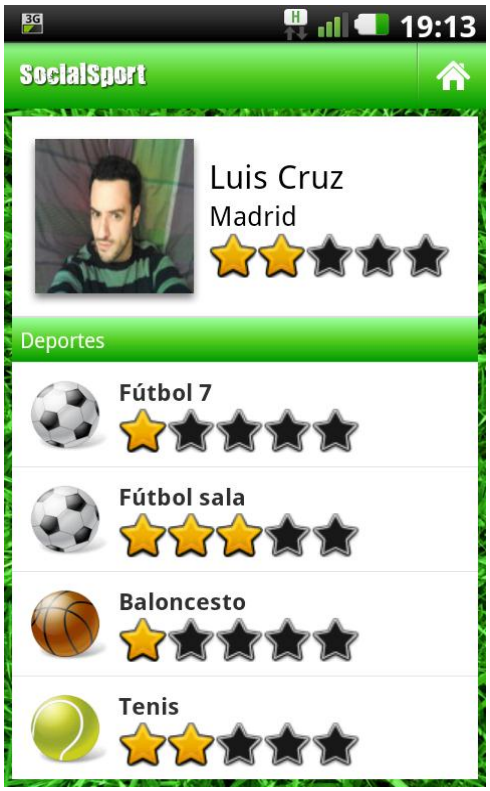
Figura 50. Menú principal con novedades

Novedades

En determinadas secciones se advierte de las novedades. Por ejemplo, si existe algún mensaje nuevo o alguna invitación a un partido, al lado del icono de acceso a la sección correspondiente podremos advertir una notificación de carácter circular y en color rojo, con el número de novedades, en cada caso (Figura 50).

Perfil

Al pulsar este botón el usuario accederá a su perfil público (Figura 51 y Figura 52). En él se mostrara: su imagen; su nivel medio deportivo, mediante una escala de estrellas; los partidos en los que forme parte, con una breve descripción del mismo; los grupos a los que pertenezca, también con algún dato descriptivo; y los deportes que le interesan con su valoración de nivel propia, medido en la misma escala que la media.



Figuras 51. Perfil público de usuario

Figura 52. Perfil público de usuario

Gente

Escogiendo esta opción, se accede a una pantalla dividida en 3 pestañas con listas, que contiene los perfiles de nuestros amigos, de todos los usuarios, y de las solicitudes que tengamos pendientes, respectivamente. Estas opciones las desarrollaremos en este mismo punto más adelante.

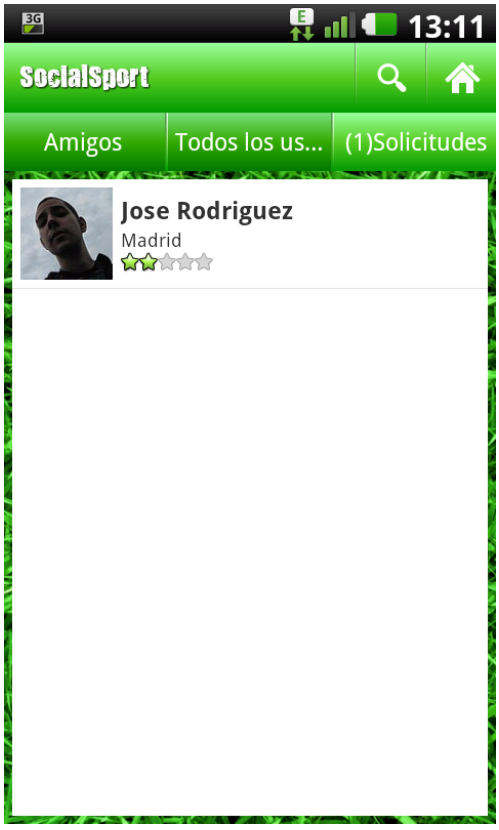


Figura 53. Solicitud de amistad

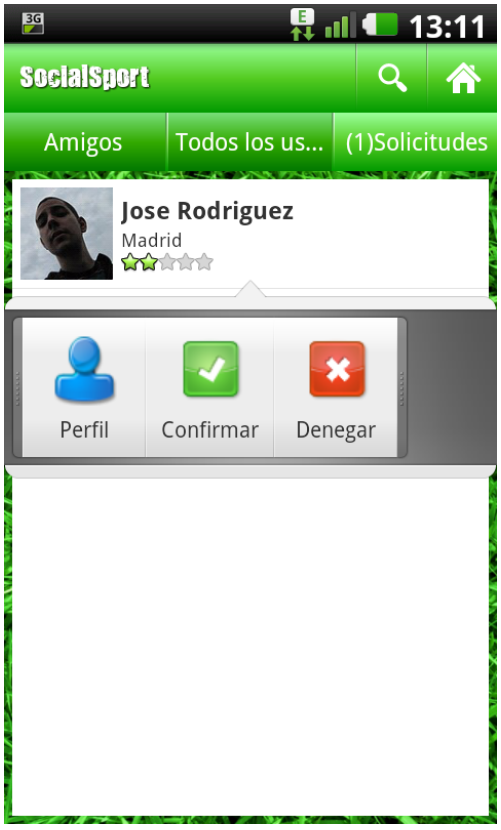


Figura 54. Opciones de una solicitud de amistad

Al lado del icono de regreso al menú principal, podemos encontrar una "lupa" que nos facilita la búsqueda de un usuario sobre la vista que esté activa en ese momento.

-Pestaña "Amigos": Se abre por defecto al acceder. Muestra la lista de personas que han aceptado la solicitud de amistad enviada y a aquellos que han sido aceptados por el usuario tras recibir la petición. Pulsando en cada elemento, se abre la opción de mostrar el perfil correspondiente.

-Pestaña "Todos los usuarios": Muestra la lista de todos los usuarios registrados en ese momento en SocialSport, que no son "amigos". Pulsando sobre cada elemento de la lista, además de la opción de acceder al perfil de usuario, se abre la posibilidad de

enviar una solicitud de amistad.

-Pestaña "Solicitudes": Muestra la lista de peticiones de amistad recibida por el usuario. Pulsando sobre cada elemento de la lista, se podrá aceptar o rechazar dicha petición (Figura 53 y Figura 54).

Grupos

Escogiendo esta opción, se accede a una pantalla dividida en dos pestañas con listas, que contienen los grupos a los que pertenece el usuario y las invitaciones de incorporación a uno que haya recibido, respectivamente. Estas opciones las desarrollaremos en este mismo punto más adelante.

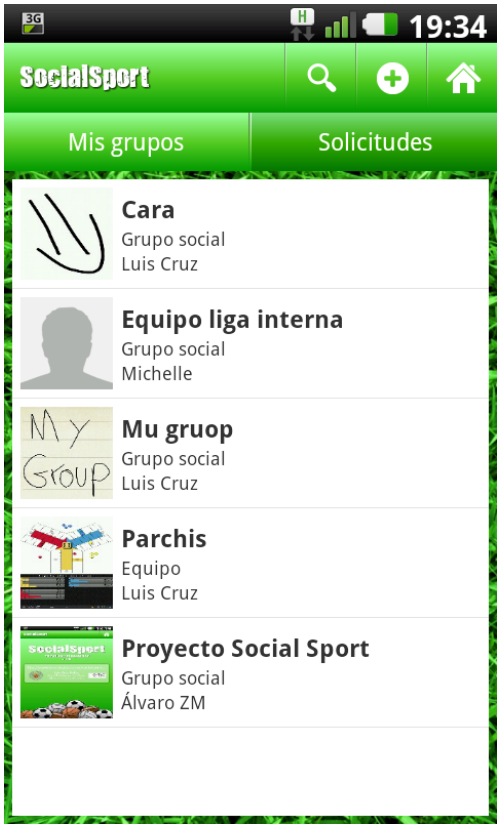


Figura 55. Lista de grupos de un usuario

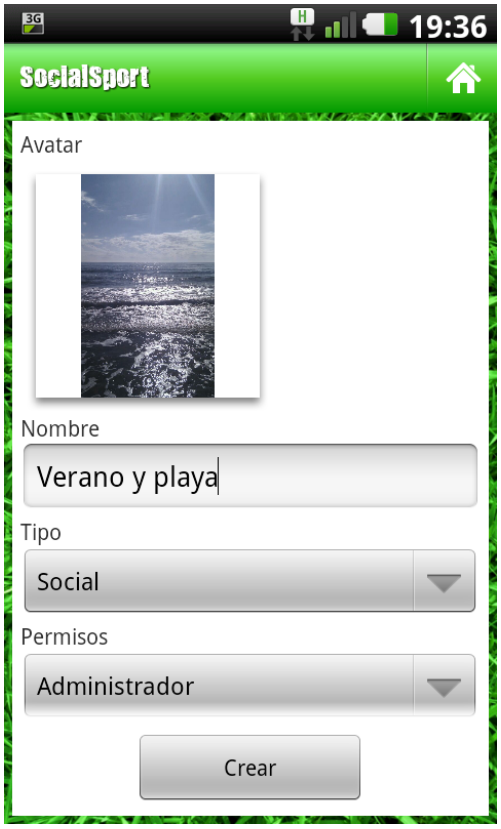


Figura 56. Creación de nuevo grupo

-Pestaña "Mis grupos": Muestra la lista de todos los grupos a los que pertenece un usuario (Figura 55). Al realizar una pulsación sobre un elemento de la lista, se accede al

grupo correspondiente.

-Pestaña "Solicitudes": Muestra la lista de peticiones de incorporación a un grupo recibida por el usuario. Pulsando sobre cada elemento de la lista, se accede al grupo correspondiente.

Nuevo grupo

Situado en la esquina superior derecha de la pantalla, se puede encontrar un icono circular con forma de "+" (Figura 57), que sirve para crear un nuevo grupo, y que también se puede encontrar sobre una vista si esta está vacía.

Al pulsarlo, se abre la configuración que se desea asignar al mismo. No todos los campos son obligatorios. Algunos vienen con un valor por defecto, pero en la mayoría de los casos es preciso que sean modificados (Figura 56). Los campos disponibles son los siguientes:

-Avatar: Permite establecer una imagen de perfil grupal, ya sea desde la cámara del dispositivo o desde la galería. No es un campo obligatorio, pero si al crear un grupo no se incorpora una, no se podrá hacer más adelante.

-Nombre: Es un campo obligatorio que representa el nombre del grupo que va a ser creado.

-Tipo: Desplegable que permite escoger entre las opciones "Equipo" y "Social". Esto permite al creador distinguir el carácter que va a tener el nuevo grupo que se cree.

-Permisos: Desplegable que permite seleccionar al usuario que cree el grupo si desea ser administrador único o que todo aquel que se incorpore al mismo también lo sea.

Para que los cambios surtan efecto y el grupo se cree, se debe pulsar el botón "Aplicar", situado en la parte inferior de la pantalla. Una vez terminado el proceso, el partido estará disponible en la pestaña "Mis grupos".

Figura 57. Botón para crear un nuevo grupo

Características y/o modificación de un grupo

Pulsando sobre un grupo disponible en cualquiera de las pestañas ya mencionadas, accedemos a la información y opciones que ofrece (Figura 58). Una vez dentro, se dispone de lo siguiente:

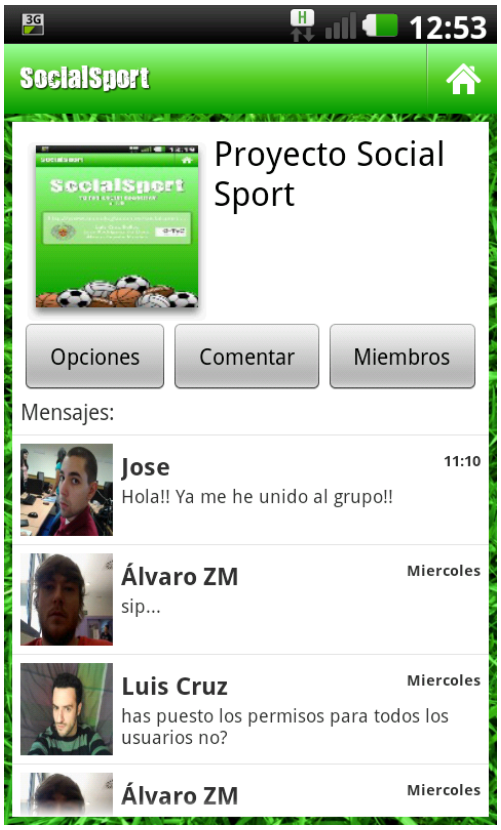


Figura 58. Perfil grupal de SocialSport

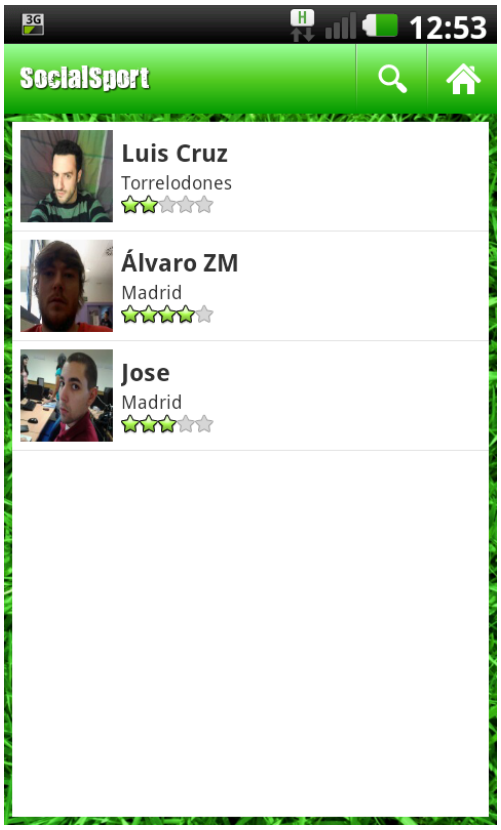


Figura 59. Lista de miembros de un grupo

-Miembros: Al presionar este botón, un una nueva vista, se listan todos los usuarios ya inscritos al grupo concreto (Figura 59).

-Comentar: Esta opción sólo está habilitada para miembros del grupo. Su funcionalidad consiste en enviar un mensaje al mismo que podrá leer todo aquel usuario que tenga acceso al grupo, ya sea por haber sido invitado o por pertenecer al mismo.

-Opciones: Las posibilidades varían dependiendo del rol del usuario. Si se trata de una invitación, las únicas opciones serán “Invitar” y “Unirse”, que permiten enviar una invitación a otro usuario para ese grupo e incorporarse al mismo, respectivamente. La primera aparecerá siempre, independientemente del rol del usuario

Por otro lado, si se pertenece a un grupo como administrador, además de eliminar el grupo pulsando la opción de nombre idéntico, se pueden añadir/eliminar administradores.

Si se es miembro de un grupo, pero no administrador, el usuario puede salirse del mismo pulsando sobre "Causo baja".

Debe tenerse en cuenta que toda modificación será instantánea, esto es, que en el momento que se seleccione se realizará, sin necesidad de pulsar algún tipo de botón que guarde los cambios de manera global.

Reservas

Esta opción permite realizar nuevas reservas de instalaciones, así como mostrar las ya existentes o cancelarlas. Inicialmente aparecerá la lista de las mismas realizadas por el usuario (Figura 60).

Nueva reserva

Para realizar una nueva reserva hay que pulsar el icono que representa una lupa, que se puede encontrar en la parte superior derecha de la pantalla, o si no hay ninguna reserva existente, también en el medio de la misma. Se abrirá una nueva vista en la que tendremos que rellenar campos tales como el deporte, la fecha o la preferencia horaria (que admite múltiple selección), con el objetivo de fijar criterios (Figura 61). Para que la búsqueda surta efecto debe pulsarse el botón aplicar. Entonces se muestra una lista expansible, ordenada por intervalos horarios. Al escoger una de ellas, se despliegan las diferentes opciones que existen para reservar (Figura 62). Se selecciona una de ellas, y aparece una última vista en la que se muestran los datos de la reserva y en la que hay que fijar la tarifa que corresponde, desde el menú desplegable que ahí se encuentra. Sólo queda pulsar el botón "Reservar", situado en la parte inferior, para que los cambios se guarden en el sistema.

Figura 60. Lista de reservas de un usuario

Figura 61. Búsqueda de reserva

Figura 62. Resultados de una búsqueda de reserva

Cancelar reserva

Otra posibilidad es cancelar una reserva. Para ello, es preciso presionar sobre el elemento de la lista, que está ordenada por proximidad temporal, que se quiere anular. Una vez se haya accedido a ella, en la parte inferior se encuentra un botón denominado "Cancelar reserva". Al pulsarlo, sale un aviso de cancelación, que es preciso confirmar para que la reserva desaparezca del sistema.

Comprobar reserva

Si simplemente se desean visualizar los datos de una reserva ya existente, es suficiente con realizar una pulsación sobre ella, y se abrirá toda la información correspondiente (Figura 63).

Figura 63. Información de una reserva

Partidos

Mediante esta opción del menú, se accede a una pantalla dividida en 3 pestañas con

listas, que contienen los partidos en los que el usuario forma parte, los partidos públicos, y las invitaciones a eventos que se hayan recibido, respectivamente.

Figura 64. Lista de partidos públicos

Figura 65. Nuevo partido

-Pestaña "Mis partidos": Se abre por defecto al acceder (Figura 68). Muestra la lista de partidos en los que el usuario forme parte, como creador o como simple participante.

-Pestaña "Públicos": Muestra la lista de partidos públicos existentes (Figura 64). Sólo se mostraran los de aquellos deportes que el usuario tenga entre sus intereses (en Ajustes->Intereses).

-Pestaña "Invitaciones": Muestra la lista de partidos a los que el usuario ha sido invitado personalmente (Figura 66). No es posible recibir invitaciones de un deporte que no se tenga seleccionado en intereses (Ajustes->Intereses).

En los 3 casos de pestañas, se accede a las características y posibilidades de un partido pulsando sobre él.

Nuevo partido

Situado en la esquina superior derecha de la pantalla, al lado de la "lupa" ya mencionada en este apartado, se puede encontrar un icono circular con forma de "+", que sirve para crear un nuevo partido, y que también se puede encontrar sobre una vista si esta está vacía (Figura 66).

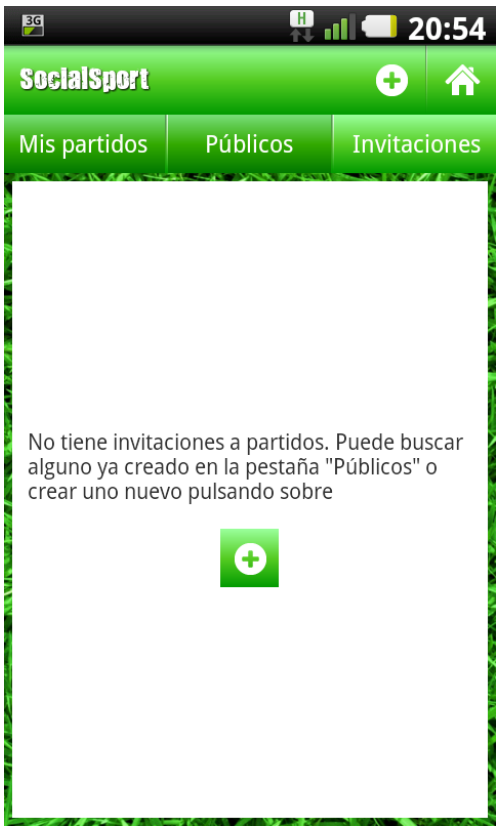


Figura 66. Vista vacía de invitaciones a partidos

Al pulsarlo, se abre la configuración que se desea asignar al mismo. Todos los campos son obligatorios. Algunos contienen un valor por defecto, pero en la mayoría de los casos es preciso que sean modificados (Figura 65). A continuación se describen los campos:

-Deporte: Pulsando sobre la figura y aparece una lista desplegable con todos los deportes existentes. Un usuario sólo puede crear un partido de un deporte que tenga previamente seleccionado entre sus intereses (en Ajustes).

-Día: Campo que viene completo por defecto con el día actual. Su función es fijar el día del partido. Para modificarlo, es preciso pulsar el botón "Establecer

día".

-Hora: Campo que viene completo por defecto con la hora actual. Su función es fijar el día del partido. Para modificarlo, es preciso pulsar el botón "Establecer hora".

-Lugar: Campo de texto en el que el usuario debe introducir, como cadena de caracteres, dónde quiere organizar el partido.

-Tipo: Desplegable en el que el usuario debe escoger si el partido es de carácter amistoso o es un partido oficial.

-Plazas: Campo que viene completo por defecto con la hora actual. Su función es fijar el número de participantes. Para modificarlo, es preciso pulsar el botón "Establecer plazas".

-Público: Casilla que determina si un partido es público, es decir, visible por todos los usuarios afines a ese deporte, o privado, mediante invitaciones personales.

Para que los cambios surtan efecto y el partido se cree, se debe pulsar el botón "Aplicar", situado en la parte inferior de la pantalla. Una vez terminado el proceso, el partido estará disponible en la pestaña "Mis partidos".

Características y/o modificación de un partido

Pulsando sobre un partido disponible en cualquiera de las pestañas ya mencionadas, accedemos a la información y opciones que ofrece (Figura 67).

En la parte superior de la pantalla se encuentran todos los datos del partido concreto: deporte, fecha, lugar y número de plazas libres.

En la zona correspondiente al medio de la pantalla aparecen las opciones que tiene el usuario:

-Opciones: Aquí, dependiendo del rol del usuario, encontramos varias posibilidades. Si es el creador, además de cancelarlo, podrá modificar la hora, el lugar, el tipo y su privacidad, seleccionando sobre "Modificar" y "Cancelar partido respectivamente". Si no es creador, puede unirse o causar baja, dependiendo de la situación concreta, y seleccionando la opción correspondiente.

Comunes a ambos roles están las opciones "Invitar" y "Acompañantes". Con la primera, es posible que usuarios que no son creadores del evento puedan

invitar a sus amigos, independientemente de que también lo sean de quien organizó el partido. La segunda permite incluir jugadores no registrados en SocialSport a la hora de buscar participantes. Debe tenerse en cuenta que los acompañantes son responsabilidad del usuario que los incluya, y que serán tenidos en cuenta tanto a la hora de contabilizar plazas, como a la de darse de baja si quien les ha inscrito se da de baja.

Figura 67. Perfil de un partido

Figura 68. Lista de partidos de un usuario

-Comentar: Presionando este botón se abre la posibilidad de enviar un comentario que podrá ser leído por todo aquel que tenga acceso al partido. Para ello, aparecerá una nueva pantalla con un campo de texto donde escribir el mensaje y un botón que, tras pulsarlo, lo publicará. Es preciso conocer que un usuario no inscrito o no invitado a un partido no puede hacer comentarios en el mismo.

-Jugadores: Al pulsar este botón aparece una lista con los jugadores registrados en SocialSport que están apuntados al partido.

Debe tenerse en cuenta que toda modificación será instantánea, esto es, que en el

momento que se seleccione se realizará, sin necesidad de pulsar algún tipo de botón que guarde los cambios de manera global.

Mensajes

Esta funcionalidad sirve para establecer y mantener una conversación privada con tus amigos. Al seleccionar esta opción, aparecerá una lista formada por todos los miembros con los que ya se haya iniciado una conversación, que incluye la imagen de perfil, el nombre de usuario, la hora del último mensaje y su contenido (Figura 69).

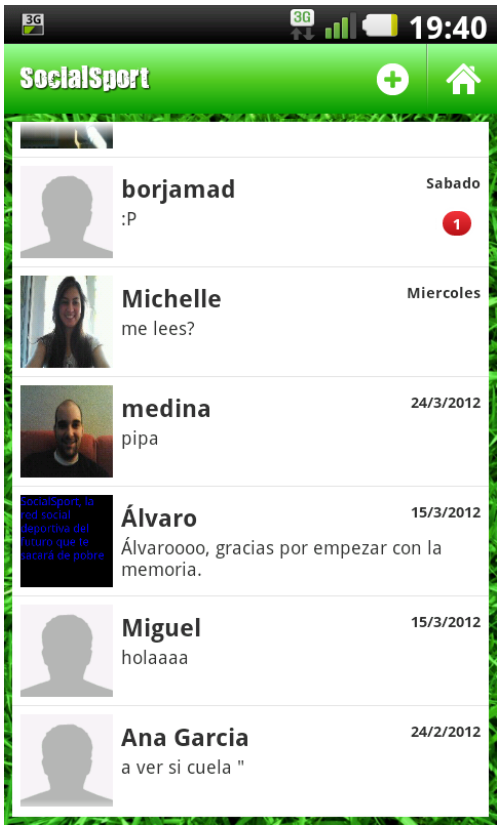


Figura 69. Lista de conversaciones de un usuario

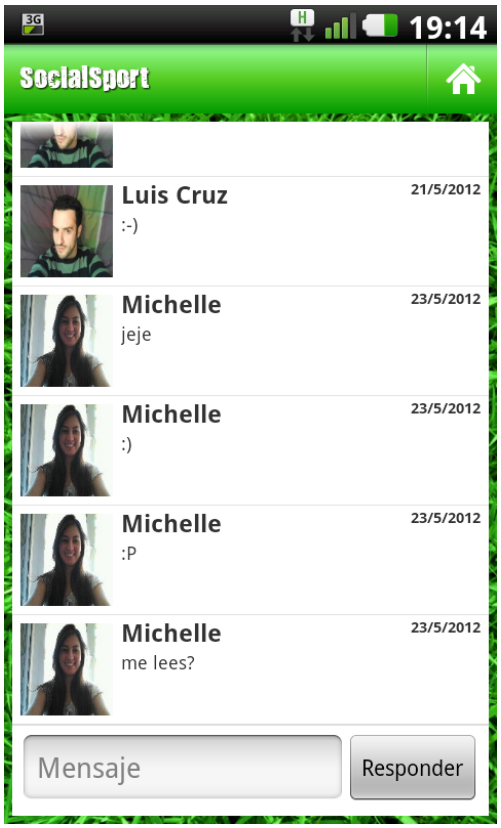


Figura 70. Ejemplo de conversación privada

Para continuar con una conversación, se pulsa sobre el elemento de la lista, y se visualizan todos los mensajes mantenidos hasta el momento, ordenados de más antiguo a más reciente. En la parte inferior de la pantalla, hay un campo de texto que permite escribir lo que se desee e inmediatamente a su derecha, el botón "Responder", que ha de ser pulsado para que el mensaje se añada a la conversación

existente (Figura 70).

Nueva conversación

Para llevar a cabo esta tarea, es preciso pulsar sobre el icono circular existente en la parte superior derecha de la pantalla, cuyo contenido es un símbolo "+", o en el medio de la misma si la vista es vacía. Entonces aparece una lista de todos los amigos del usuario. Se puede buscar usuarios registrados presionando en el icono con forma de "lupa" de la parte superior derecha de la pantalla. Pulsando sobre el deseado se tiene la posibilidad de enviar un mensaje nuevo, iniciando así una conversación. Si la pulsación se produce sobre alguien con quien ya se había establecido una cadena de mensajes con anterioridad, simplemente se añade y continúa normalmente. Para ello, basta con rellenar el campo de texto con lo deseado y pulsar el botón de la parte inferior de la pantalla.

Información

Muestra la información de la aplicación, con el nombre de la misma, la versión, los desarrolladores, un enlace de referencia, y los iconos de la universidad y el grupo de investigación y desarrollo a cargo del proyecto (Figura 71).

En esta vista no es posible realizar ningún tipo de edición.

Ajustes

A través de esta sección se podrán realizar las modificaciones relativas a los datos personales del usuario (Figura 72 y Figura 73). Debe tener en cuenta que, algunos campos, como por ejemplo la imagen de "Avatar" o el nombre de usuario, serán visibles a cada miembro registrado de SocialSport. Inicialmente, esta imagen de perfil ya mencionada, los datos de localización o las preferencias deportivas del usuario estarán y permanecerán vacíos hasta que los rellene manualmente.

Figura 71. Información de la aplicación

Para que todos los cambios realizados sean guardados, debe pulsarse el botón "Aplicar", situado en la parte inferior de la pantalla. De no ser así, toda modificación se perderá.

-Imagen de perfil: Pulsando en el icono existente bajo el texto "Avatar", se podrá cambiar la imagen de perfil, visible para todos los usuarios, ya sea tomando una nueva utilizando la cámara o seleccionándola desde la galería de fotos del dispositivo.

-Nombre de usuario: Para disponer de un nombre de usuario diferente a aquel con el que se registró, basta con rellenar el campo bajo la etiqueta "Usuario", con el que desee.

-Contraseña: Si se desea cambiar la contraseña, es suficiente con escribir la nueva en los espacios habilitados para ello, tanto bajo el texto "Contraseña", como bajo "Confirmar contraseña". En caso de no coincidencia se produce un mensaje de error. Una vez realizada correctamente esta modificación, se recomienda la inmediata reconexión (desconectarse para volver a conectarse) utilizando ya la nueva contraseña de cara a evitar posibles olvidos futuros.

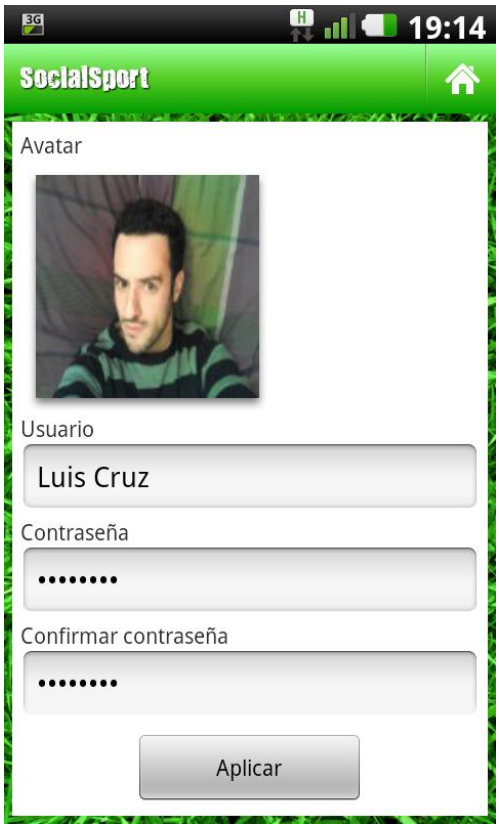


Figura 72. Ajustes de usuario

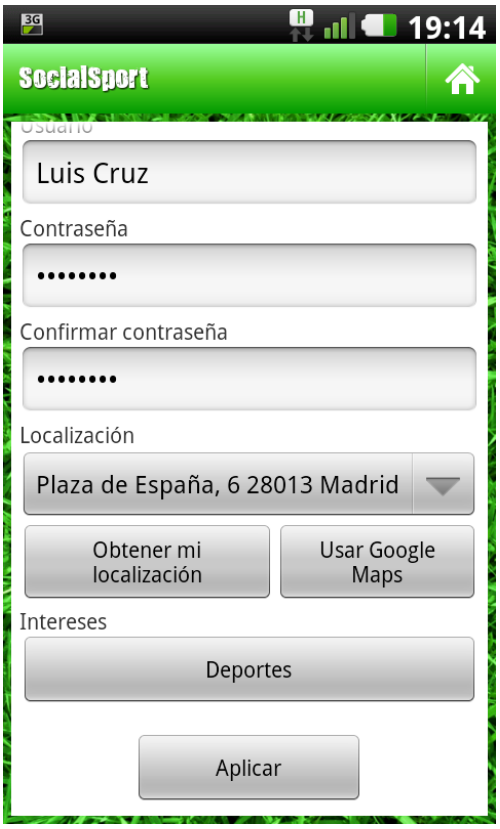


Figura 73. Ajustes de usuario

-Localización: Las ubicaciones introducidas, hasta un máximo de 10, se mostrarán en el desplegable al que se puede acceder pulsando en el campo inmediatamente inferior a la etiqueta "Localización".

Para activar una nueva ubicación, hay 3 opciones:

-Desplegar el menú ya mencionado y seleccionar una localización existente pulsando sobre ella.

-Pulsar el botón "Obtener mi localización". Automáticamente se detectará la ubicación del usuario en ese momento, y la dirección en que se encuentre se añadirá al desplegable y quedará seleccionada (por problemas de Android, a veces es preciso repetir esta operación para que se realice correctamente).

-Pulsar el botón "Usar Google Maps" (para más información, ver siguiente apartado de ésta sección).

Nótese que al obtener una nueva ubicación mediante los botones "Obtener mi localización" o "Usar Google Maps", ésta quedará registrada en el desplegable y aunque cierre o desconecte la aplicación, se guardará y estará disponible en este tipo

de menú la próxima vez que acceda.

Por otro lado, si se tienen ya 10 ubicaciones pertenecientes a un mismo perfil, y se introduce una nueva, superando por tanto el límite, se eliminará la primera que se introdujo, desapareciendo completamente del menú desplegable, desde el momento en que se apliquen los cambios.

Usar Google Maps

Escogiendo esta opción se puede especificar una ubicación en cualquier lugar del planeta. Para ello, hay que pulsar en el icono con forma de lupa de la parte superior derecha de la pantalla. Aparece un campo donde introducir la dirección deseada y un botón denominado "Buscar" que se debe pulsar para realizar una búsqueda (Figura 74). Este campo desaparece si se vuelve a presionar sobre el icono mencionado. Una vez llevados a cabo estos pasos, existen 3 posibilidades:

-Que la dirección introducida sea incorrecta, por lo que se mostrará un mensaje por pantalla advirtiendo de que se concrete más la búsqueda.

-Que sea ambigua. Aparecerá una lista con 5 opciones y se deberá seleccionar aquella que se desee. Si ninguna de ellas es satisfactoria, se recomienda realizar una búsqueda más concreta.

-Que sea correcta.

Cuando se haya obtenido una dirección correcta, el mapa se situará en la ubicación especificada y se colocará un icono con forma de chincheta y de color verde. Una vez exista este tipo de marcador, también se puede obtener nuevas localizaciones pulsando sobre el mapa. La dirección que se haya "tocado" se mostrará en el campo de búsqueda, y si se quiere situar el marcador con forma de chincheta ahí, es suficiente con pulsar el botón "Buscar".

Para que los cambios tengan efecto y la nueva dirección aparezca en el menú desplegable de localizaciones que encontramos en la sección "Ajustes", debe pulsarse el botón "Aceptar". Debe tenerse en cuenta que si se aplican los cambios antes de que se haya realizado una búsqueda con éxito (o lo que es lo mismo, antes de que exista un marcador como el ya mencionado en algún lugar del mapa), el sistema no tendrá nada que guardar y por lo tanto el desplegable no se actualizará.

Pulsando el botón "Cancelar", no solo no se guardarán los cambios realizados, sino que se volverá a la pantalla de "Ajustes".

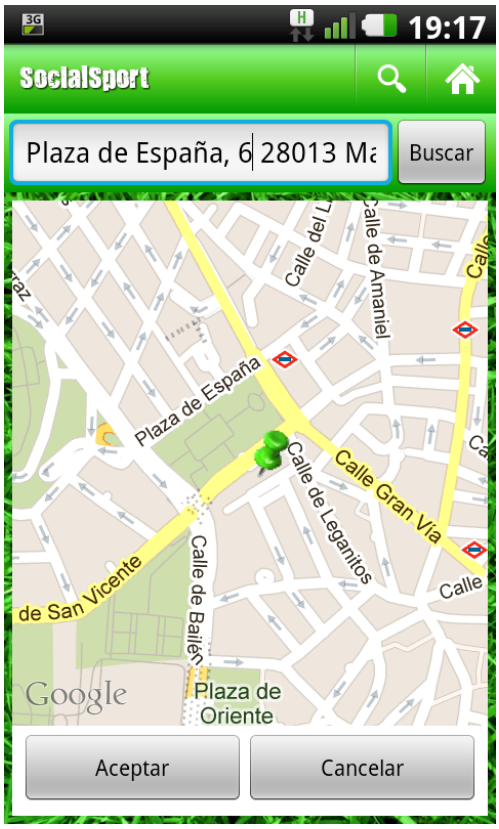


Figura 74. Google Maps en SocialSport

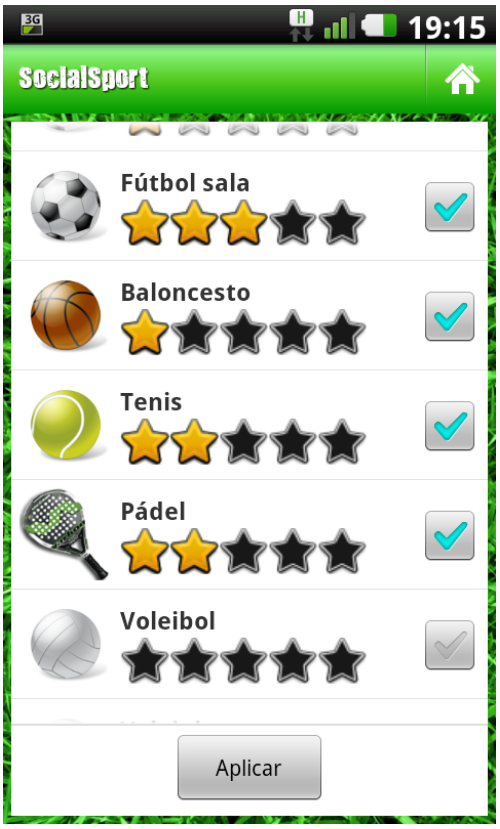


Figura 75. Deportes interesantes

-Intereses: Presionando este botón se accede a la lista de deportes disponibles.

Intereses

Aquí, se pueden seleccionar aquellos deportes que resulten interesantes, pulsando en la parte derecha o izquierda de la zona relativa a cada uno, para que quede constancia (Figura 75). Esto servirá para que no lleguen notificaciones y eventos que no interesen al usuario, siendo más cómodo para él. No obstante, sí podrá crearlos independientemente de la selección que tenga.

Por otro lado, aproximadamente en el medio del espacio reservado a cada deporte, se pueden encontrar unas estrellas a modo de "evaluador de nivel". El jugador debe asignarse el valor que estime oportuno para que el resto de usuarios pueda conocer el nivel aproximado que tiene en cada deporte.

Para salir de esta pantalla y que se guarden los cambios realizados es preciso presionar el botón denominado "OK" que aparece en la parte inferior.

Desconectar

Se podrá realizar la desconexión completa de la aplicación accediendo a esta opción

desde el menú principal, y pulsando sobre el icono denominado “Desconectar”, situado en la esquina inferior derecha de la pantalla.

La cuenta de SocialSport se mantendrá conectada desde el dispositivo en que se esté ejecutando a no ser que se lleve a cabo la opción expuesta en este apartado.

Figura 76. Botón desconectar

CONCLUSIONES Y TRABAJO FUTURO

Conclusiones

SocialSport es una aplicación muy completa, que abarca la funcionalidad de redes sociales y organización de eventos deportivos. Por un lado, permite conocer gente con inquietudes similares, formar parte de un equipo o grupo o comunicarte con el resto de usuarios de manera directa; y por otro, facilita muchísimo la labor de organizar o participar en eventos deportivos, pudiendo escoger entre un amplio rango de configuraciones diferentes.

Incluye también las bases necesarias para que la aplicación evolucione hacia una herramienta de reserva de instalaciones no simulada, lo que abre bastante las posibilidades de mercado, ya que agilizaría los trámites.

Además, con el sistema de notificaciones, el usuario estará avisado en tiempo real, a través de su dispositivo, de las novedades que vayan surgiendo sobre su persona, lo que agiliza el proceso de inclusión en un evento o mismamente de recibir una información concreta.

Nuestra intención, como equipo de desarrollo es completar esta herramienta para que pueda ser una realidad en el mercado actual. Para ello, hemos establecido conexiones con empresas que puedan proporcionar financiación, y con clientes potenciales interesados en la adaptación de SocialSport para su comercialización y explotación en un futuro próximo.

Líneas de trabajo futuro

Si todo se desarrolla según lo previsto, SocialSport seguirá evolucionando en los próximos años, ofreciendo nuevas funcionalidades, y completando y mejorando las ya existentes. El objetivo es seguir creciendo de cara a conseguir una aplicación óptima, con el objetivo de ayudar a cada vez más gente, a practicar sus deportes favoritos sin problemas.

Una de las funcionalidades a desarrollar es la de poder incluir un buscador de eventos y personas más potente, que permita seleccionar entre varias opciones, y que muestre los resultados por proximidad. De este modo, un usuario podría realizar búsquedas configuradas a su gusto, o simplemente que la aplicación le recomendara eventos acordes a sus características individuales o jugadores que cuyo perfil se ajuste a la configuración establecida de un evento.

Relacionado con esto, sería muy interesante que a la hora de crear un evento, la propia aplicación mostrara una lista de instalaciones que se ajustaran a las exigencias establecidas, teniendo también en cuenta la ubicación de los participantes.

También estamos interesados en realizar la gestión de torneos de diferentes tipos, basado en perfiles grupales (equipos). Esta funcionalidad requeriría un control exhaustivo de los equipos apuntados, para que existiera la máxima fiabilidad y compromiso de los inscritos. Para ello, el perfil grupal debería ser más completo y poder interactuar con él.

A nivel individual, pretendemos ampliar las posibilidades incorporando nuevos tipos de perfil, como pueden ser las figuras de árbitro o entrenador, además de la de jugador ya existente. También nos gustaría poder realizar evaluaciones personales tras la disputa de un evento, de modo que sea el resto de usuarios quien evalúe el nivel, en vez de hacerlo uno mismo. En cuanto a los grupos, una funcionalidad interesante para el futuro es el de utilizarlos como equipos en la gestión de un torneo.

Al intentar obtener la ubicación actual del usuario, en ocasiones es preciso que el proceso deba repetirse para que se realice satisfactoriamente. Esto se debe a que, como se trata de un proceso que puede resultar lento, Android devuelve la última ubicación conocida. De hecho, este problema también era bastante común en la famosa aplicación WhatsApp©, que devolvía la última ubicación conocida al realizar la petición, en vez de la actual. Este problema se tendrá en cuenta a la hora de seguir desarrollando.

Hay elementos que actualmente están concebidos como meramente informativos, pero que se han creado pensando en que sirven para completar y aumentar funcionalidades. Se podría decir que son partes a las que en un futuro se les dará un uso más apropiado e interesante del que tienen en la actualidad. Los ejemplos más claros son la ubicación y la valoración de nivel por deporte.

Un aspecto a tener en cuenta es el número de usuarios. Además de la continua publicación de versiones mejoradas, la distribución de la aplicación en nuevos mercados más propicios a su consumo ayudará a conseguir un mayor número de personas registradas.

Por otro lado, también se espera tener un portal web de calidad, que dé soporte a la aplicación, donde el usuario tendría acceso a todas las funcionalidades ya disponibles. Esto sería importante de cara a la captación de usuarios y a la comodidad a la hora de manejar este servicio.

REFERENCIAS

Bibliografía

- Ed Burnett; *Hello, Android: Introducing Google's Mobile Development Platform*; Pragmatic Bookshelf, 2010.
- Frank Ableson, Robi Sen, Chris King; *Android, guía para desarrolladores*; Segunda edición; Anaya Multimedia, 2011.
- Joan Ribas Lequerica; *Desarrollo de aplicaciones para Android*; Anaya Multimedia, 2011.
- Jeff Friesen, *Java para desarrollo Android*; Anaya Multimedia, 2011.

Enlaces bibliográficos

Instalación del SDK de Android:

- http://developer.android.com/sdk/installing.html

Instalación del plugin ADT para Eclipse:

- http://developer.android.com/sdk/eclipse-adt.html

Desarrollo en Android:

- http://developer.android.com/index.html

Complemento al desarrollo en Android:

- http://www.sgoliver.net/blog/?p=1313
- http://www.vogella.com/

Foros de resolución de dudas y problemas:

- http://stackoverflow.com/
- http://code.google.com/

Creación de layouts de Android:

- http://mobile.tutsplus.com/tutorials/android/android-layout/

Subir imagen a un servidor

- http://www.coderzheaven.com/2011/04/25/android-upload-an-image-to-a-server/

Notificaciones Android (Notification Bar):

- http://blog.findemor.es/2011/10/programar-en-android-notificaciones/

Notificaciones PUSH:

- http://tokudu.com/2010/how-to-implement-push-notifications-for-android/

Bubble Notifications (Notificaciones dentro de la aplicación)

- http://blog.donnfelker.com/2011/12/02/building-an-android-notification-bubble/

Iconos aplicación:

- http://envyandroid.com/archives/271/easiest-way-to-create-android-icons

Manejo de la clase “Service” en Android:

- http://developerlife.com/tutorials/?p=356

Ubicación en Android:

- http://www.maestrosdelweb.com/editorial/curso-android-geolocalizacion-utilizacion-mapas-google/

Implantación de Google Maps:

- http://www.codeproject.com/Articles/112044/GPSLocator-App-to-Find-Current-Nearest-Location-us

Diferentes funcionalidades usando Google Maps:

- http://mobiforge.com/developing/story/using-google-maps-android

ANEXO: MANUAL DE ANDROID

Introducción a Android

Manuel Báez, Álvaro Borrego, Jorge Cordero,
Luis Cruz, Miguel González, Francisco Hernández,
David Palomero, José Rodríguez de Llera, Daniel Sanz,
Mariam Saucedo, Pilar Torralbo, Álvaro Zapata

G-TeC
www.tecnologiaUCM.es

Introducción a Android
ISBN: 978-84-96285-39-5
E.M.E. Editorial ©
Editores: Victoria López (vlopez@fdi.ucm.es) y Grupo Tecnología UCM (www.tecnologiaUCM.es)

Tabla de contenido

1. INTRODUCCIÓN

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

¿QUÉ ES ANDROID?

En los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales, grandes y pesados, pensados sólo para hablar por teléfono en cualquier parte, a los últimos modelos, con los que el término "medio de comunicación" se queda bastante pequeño.

Es así como nace Android. Android es un sistema operativo y una plataforma software, basado en Linux para teléfonos móviles. Además, también usan este sistema operativo (aunque no es muy habitual), tablets, netbooks, reproductores de música e incluso PC's. Android permite programar en un entorno de trabajo (framework) de Java, aplicaciones sobre una máquina virtual Dalvik (una variación de la máquina de Java con compilación en tiempo de ejecución). Además, lo que le diferencia de otros sistemas operativos, es que cualquier persona que sepa programar puede crear nuevas aplicaciones, *widgets*[1], o incluso, modificar el propio sistema operativo, dado que Android es de código libre, por lo que sabiendo programar en lenguaje Java, va a ser muy fácil comenzar a programar en esta plataforma.

HISTORIA DE ANDROID

Fue desarrollado por Android Inc., empresa que en 2005 fue comprada por Google, aunque no fue hasta 2008 cuando se popularizó, gracias a la unión al proyecto de Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo hardware, software y telecomunicaciones, que decidieron promocionar el software libre. Pero ha sido Google quien ha publicado la mayor parte del código fuente del sistema operativo, gracias al software Apache, que es una fundación que da soporte a proyectos software de código abierto.

Dado que Android está basado en el núcleo de Linux, tiene acceso a sus recursos, pudiendo gestionarlo, gracias a que se encuentra en una capa por encima del Kernel, accediendo así a recursos como los controladores de pantalla, cámara, memoria flash…

En la Figura 1, abajo, se muestran las capas que conforman el sistema operativo Android:

[1] Un *widget* es una pequeña aplicación que facilita el acceso a funciones frecuentes. Más información en el Capítulo 10

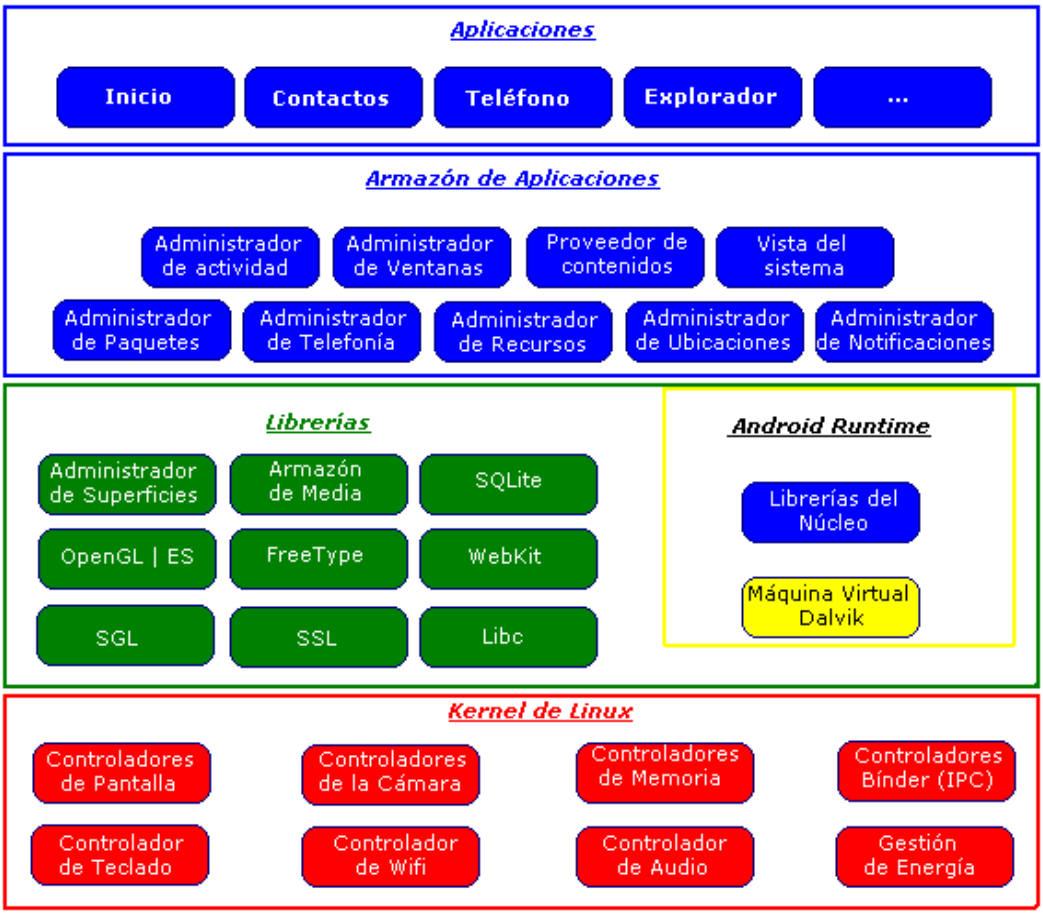


Figura 1. Sistema de capas de Android

En la imagen se distinguen claramente cada una de las capas: la que forma parte del propio Kernel de Linux, donde Android puede acceder a diferentes controladores, las librerías creadas para el desarrollo de aplicaciones Android, la siguiente capa que organiza los diferentes administradores de recursos, y por último, la capa de las aplicaciones a las que tiene acceso.

VERSIONES DISPONIBLES

El sistema operativo Android, al igual que los propios teléfonos móviles, ha evolucionado rápidamente, acumulando una gran cantidad de versiones, desde la 1.0 para el QWERTY HTC G1, hasta la 4.0 que acaba de salir al mercado.

- **Cupcake: Android Version 1.5**

  Características: Widgets, teclado QWERTY virtual, copy & paste, captura de vídeos y poder subirlos a Youtube directamente.

- **Donut: Android Version 1.6**

  Características: Añade a la anterior la mejoría de la interfaz de la cámara, búsqueda por voz, y navegación en Google Maps.

- **Eclair: Android Version 2.0/2.1**

Características: Mejoras en Google Maps, salvapantallas animado, incluye zoom digital para la cámara, y un nuevo navegador de internet.

- **Froyo: Android Version 2.2**

Características: Incluye hostpot Wifi, mejora de la memoria, más veloz, Microsoft Exchange y video-llamada.

- **Ginger Bread: Android Version 2.3**

Características: Mejoras del consumo de batería, el soporte de vídeo online y el teclado virtual, e incluye soporte para pagos mediante NFC[2].

- **Honey Comb: Android Version 3.0/3.4**

Características: Mejoras para tablets, soporte Flash y Divx, integra Dolphin, multitarea pudiendo cambiar de aplicación dejando las demás en espera en una columna, widgets y homepage personalizable.

- **Ice Cream Sandwich: Android Version 4.0**

Características: Multiplataforma (tablets, teléfonos móviles y netbooks), barras de estado, pantalla principal con soporte para 3D, widgets redimensionables, soporte usb para teclados, reconocimiento facial y controles para PS3.

ECLIPSE COMO ENTORNO DE TRABAJO

En este curso de Android, se da por supuesto que el alumno está familiarizado con el entorno Eclipse y que además tiene nociones básicas de programación en el lenguaje Java. Lo primero que necesitaremos para poder programar en Android, es preparar el entorno de trabajo. Es necesario disponer de una versión de Eclipse Galileo 3.5 o superior para poder desarrollar nuestros proyectos. Lo segundo que necesitamos es el kit de desarrollo software para Android o Android SDK, del que se pueden encontrar varias versiones para diferentes plataformas en la página web:

http://developer.android.com/sdk/index.html

Si el sistema operativo es Windows, lo más recomendable, es descargar el instalador automático **installer_rXX-windows.exe**, y simplemente seguir las instrucciones. Una vez se inicia la instalación, el instalador comprueba si el equipo dispone del Java SE Development Kit (JDK). Si no es así, muestra un mensaje como el siguiente:

[2] NFC (Near-Field Communication) es una plataforma abierta para la comunicación instantánea.

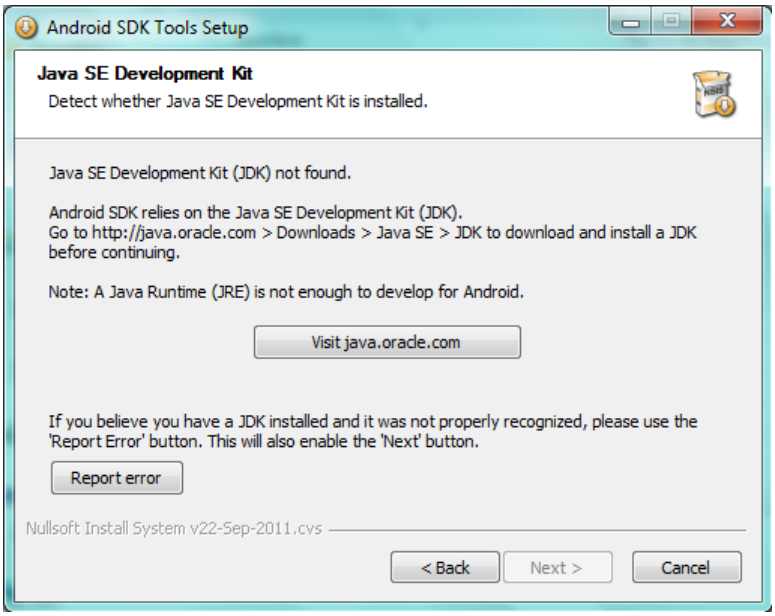


Figura 2. Android Setup

Simplemente pincha sobre el botón "Visit java.oracle.com" (Figura 1.1) que redireccionará a la página mencionada para descargar el paquete necesario. Una vez instalado el JDK, se continúa con la instalación del SDK. Cuando finalice el instalador, se ejecutará el SDK Manager, en el que se deberán seleccionar todas las casillas deshabilitadas, para instalar todas las versiones de Android así como sus herramientas (Tools).

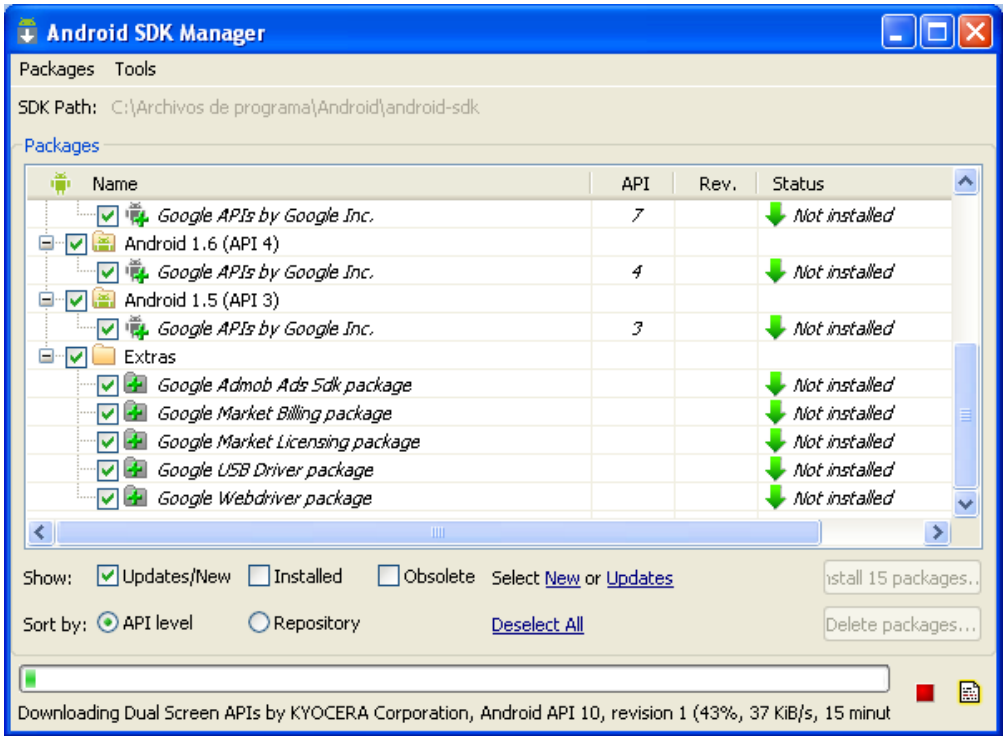


Figura 3. SDK Manager

Una vez todo esté descargado e instalado, abrir Eclipse para descargar el **ADT Plugin** e instalarlo en el entorno de desarrollo. Se deben seguir los siguientes pasos:

1.- En la pestaña “Help”, seleccionar “Install New Software”.

2.- Presionar el botón “Add” en la esquina superior derecha.

3.- En el cuadro de dialogo que aparece, escribir “ADT Plugin” en el campo “Name”, y la siguiente URL

en el campo “Location” y pulsar “OK” (Si existe algún problema para enlazar el entorno con éste link,

probar a poner http: eliminando la ‘s’):

https://dl-ssl.google.com/android/eclipse/

4.- En “Avalaible Software”, seleccionar la casilla correspondiente a “Developer Tools” y pulsar “Next”.

5.- Leer y aceptar el Acuerdo de licencia y presionar “Finish”(si salta una advertencia de seguridad

informando de que la autenticidad o validez del software no se puede establecer, simplemente pulsar

“OK”), y reiniciar Eclipse.

Lo único que queda es configurar el ADT Plugin. En Eclipse, en la pestaña “Window”, seleccionar “Preferences”, y elegir “Android” en el panel de la izquierda. Aparecerá un cuadro de dialogo preguntándo si se quiere enviar estadísticas a Google, seleccionar la elección y pulsar “Proceed”. Ahora presionar el botón “Browse” y seleccionar la ruta del directorio dónde se haya ubicado el SDK (normalmente C:\Archivos de programa\Android\Android-sdk\) y pulsar “Apply” y “OK”.

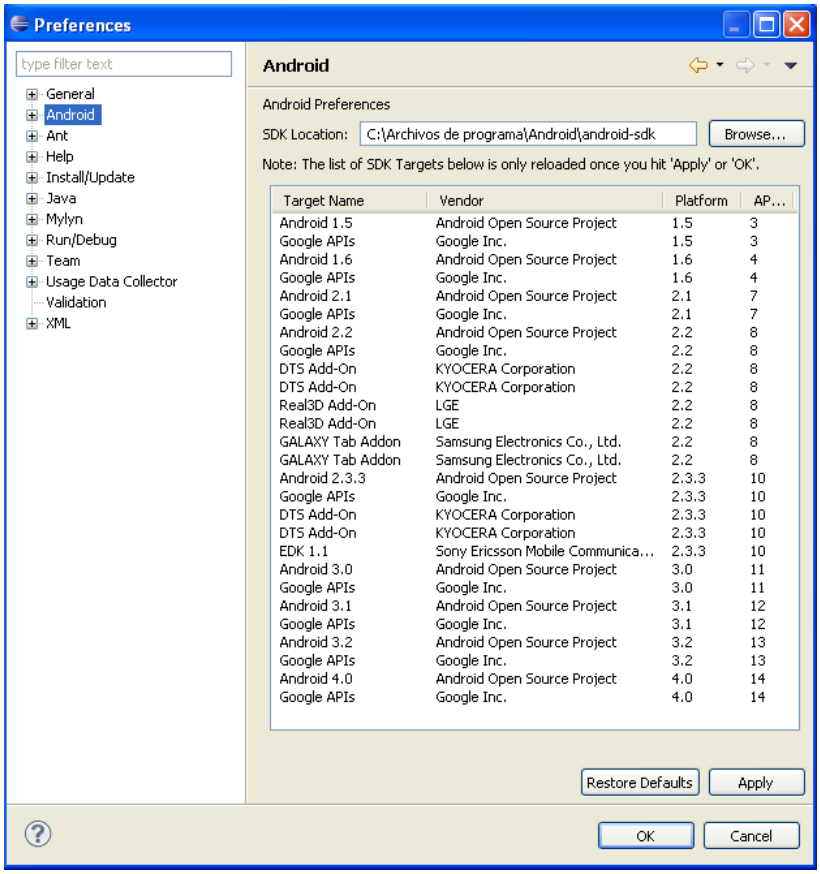


Figura 4. Preferences

Por último, hay que comprobar que el SDK está completamente actualizado. Para ello, en la pestaña “Window”, seleccionar “Android SDK and AVD Manager”. En la sección “Available

Packages", seleccionar todas aquellas casillas a instalar. Presionar "Install Selected" para comenzar con la descarga e instalación.

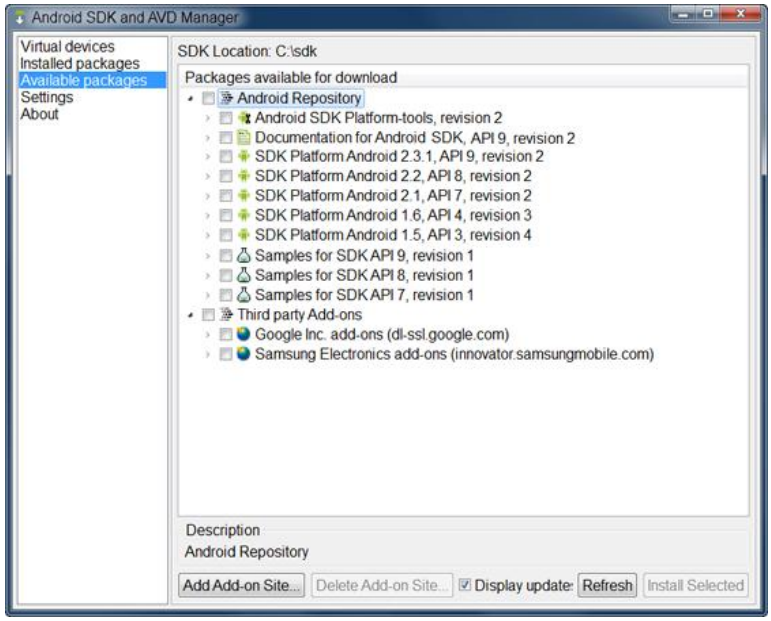


Figura 5. Repository

¡Y ya está! Ya tenemos preparado el entorno para poder programar en Android.

PERSPECTIVAS Y EMULADOR

1. PERSPECTIVA JAVA

Dados por sabidos los conocimientos básicos sobre Eclipse y la programación en Java, ésta perspectiva debe ser conocida por todos.

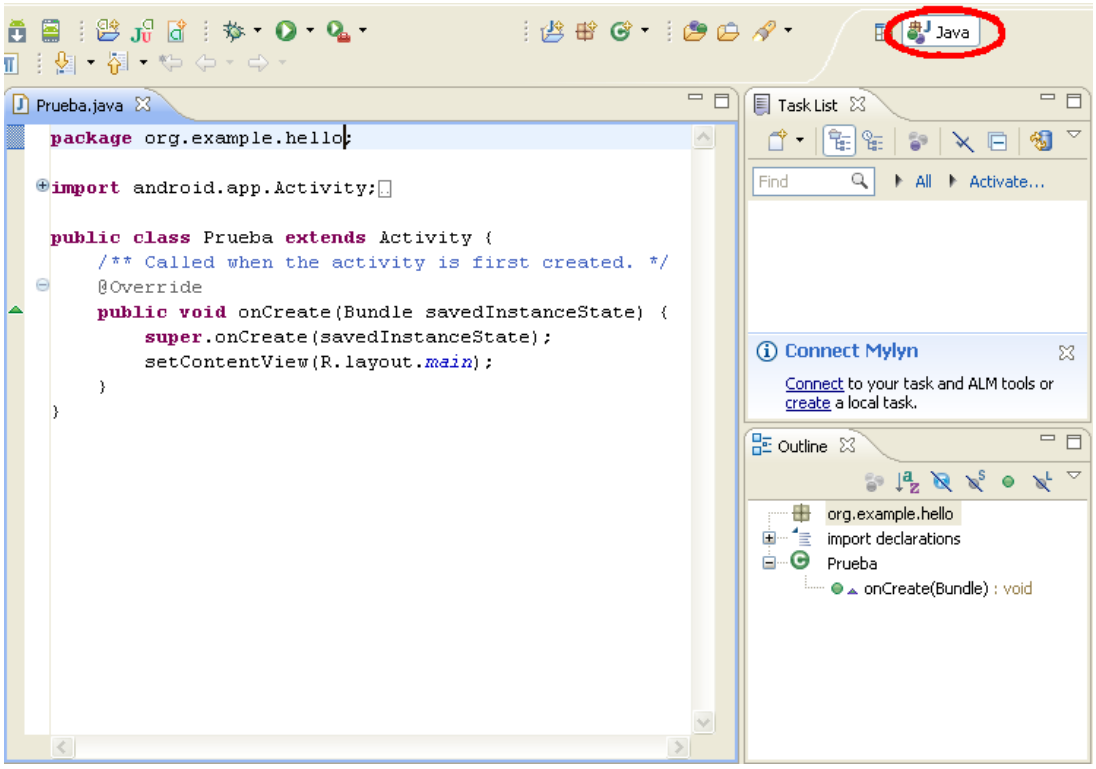


Figura 6. Perspectiva Java

Es la interfaz de usuario (o conjunto de vistas) que provee el JDT Plugin para poder programar en lenguaje Java. Esta interfaz, proporciona una serie de herramientas (se puede considerar como una determinada organización de las vistas), para el correcto desarrollo de programas y aplicaciones, y será la que utilizaremos para programar en este curso de Android.

2. PERSPECTIVA DDMS

En este caso, es el ADT Plugin el que nos proporciona la nueva perspectiva, por lo que lo primero que hay que hacer es habilitarla.

En la pestaña "Window", seleccionar "Open Perspective" -> "Other"-> "DDMS".

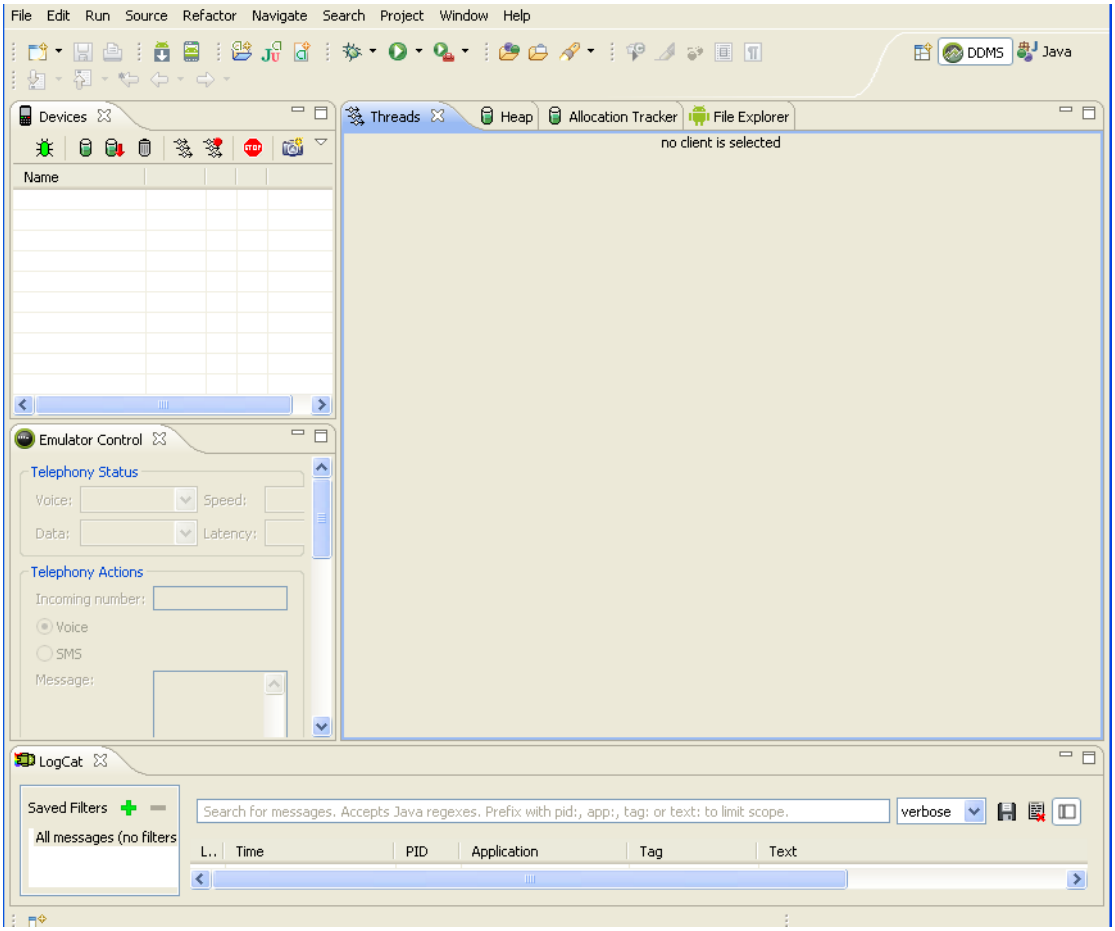


Figura 7. Perspectiva DDMS

Esta perspectiva, sirve para poder programar y realizar debugging al mismo tiempo, lo que es una forma muy efectiva de programar.

Aunque se programará con la perspectiva Java, a la hora de corregir errores se puede pasar a la perspectiva DDMS.

3. EMULADOR

Una vez tengamos el proyecto listo para ejecutar, entra en escena el emulador de Android. Éste proporciona una vista especial para comprobar si la aplicación hace lo que se desea. A continuación se muestra la vista del emulador para la versión 2.2 de Android:

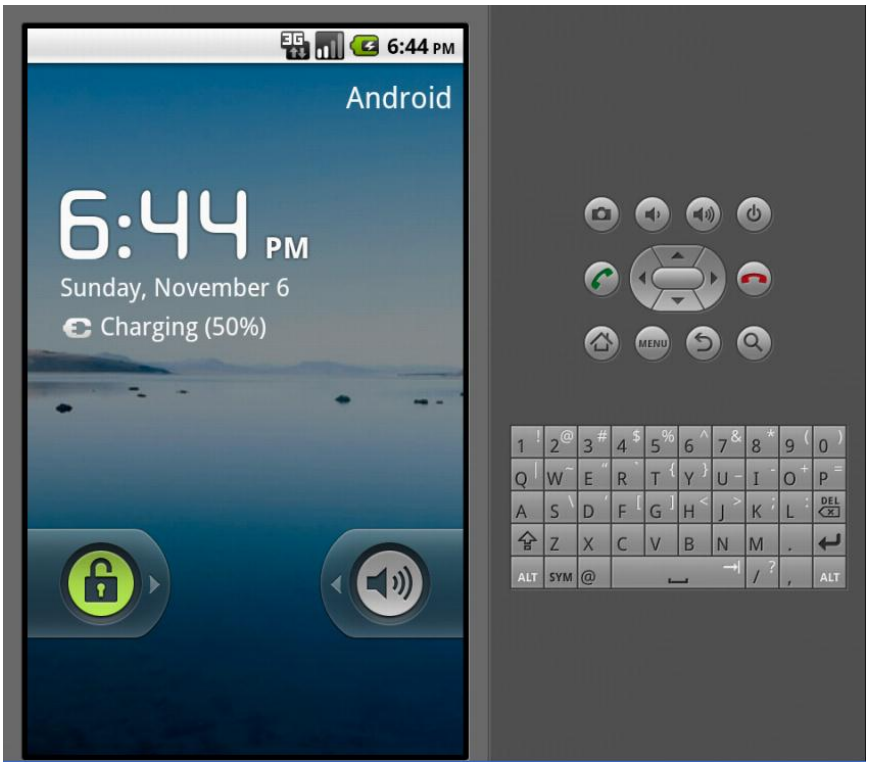


Figura 8. Emulador para Android 2.2

Lo primero que hay que hacer cuando se quiere ejecutar una aplicación, es pinchar sobre el proyecto con el botón derecho, y en "Run as" seleccionar "Android Application", entonces se lanzará el emulador más apropiado siempre que esté creado (más adelante, se explicará cómo generar los emuladores).

No se debe parar la ejecución del emulador, dado que cada vez que se ejecuta el mismo, necesita de muchos recursos del computador, por lo que tarda bastante en lanzarse, y realmente no es necesario cerrarlo, puesto que cada vez que se lleva a cabo una ejecución del proyecto, la aplicación se reinstala en el emulador.

UN EJEMPLO: "HOLA ANDROID"

Vamos a crear nuestro primer proyecto en Android, pero antes veamos de qué se compone cada uno. Al generar un nuevo proyecto de Android, dado que estamos utilizando el entorno Eclipse, éste va a generar automáticamente la distribución de carpetas que contendrá la aplicación, la cuál será común a todos los proyectos Android.

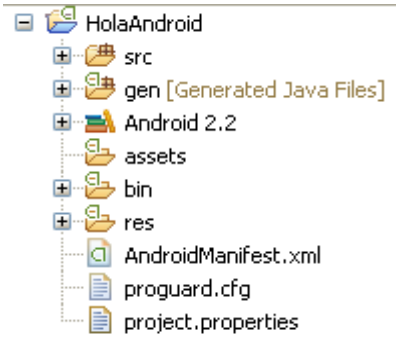


Figura 9. Sistema de carpetas de un proyecto

Veamos el significado de cada carpeta por separado:

- **Carpeta src:**

  Recoge la totalidad del código fuente (Java) de la aplicación. En el ejemplo que vamos a llevar a cabo, Eclipse generará automáticamente el código base de la ventana principal (Activity).

- **Carpeta res:**

Contiene los recursos necesarios para generar una aplicación Android:

.- res/drawable/: Guarda las imágenes y se divide en: drawable-ldpi, drawable-mdpi y drawable-hdpi, que dependerán de la resolución del dispositivo.

.- res/raw/: Contiene archivos de propósito general, en otro formato que no es XML.

.- res/layout/: Incluye los archivos que definen el diseño de la interfaz gráfica, siempre en XML.

.- res/values/: Guarda los datos y tipos que utiliza la aplicación, tales como colores, cadenas de texto, estilos, dimensiones…

- **Carpeta gen:**

  Ésta carpeta guarda un conjunto de archivos (de código Java) creados automáticamente cuando se compila el proyecto, para poder dirigir los recursos de la aplicación. El archivo R ajusta automáticamente todas las referencias a archivos y valores de la aplicación (guardados en la carpeta res).

- **Carpeta assets:**

Guarda el resto de archivos necesarios para el correcto funcionamiento de la aplicación, como los archivos de datos o de configuración. La principal diferencia entre los recursos que almacena ésta carpeta y los que guarda la carpeta "res", es que los recursos de ésta última generan un identificador por recurso, identificador que se encargará de gestionar el fichero R y sólo se podrá acceder a ellos a través de determinados métodos de acceso, mientras que los recursos almacenados en la carpeta "assets" no generan identificador alguno y se accederá a ellos a través de su ruta, como se hace con cualquier otro fichero.

- **Archivo AndroidManifest.xml:**

Éste archivo es uno de los más importantes de cualquier aplicación Android. Se genera automáticamente al crear el proyecto, y en él se encuentra definida la configuración del proyecto en XML (Actividades, Intents, los permisos de la aplicación, bibliotecas, etc.). Por ejemplo, el proyecto que vamos a generar ("Hola Android"), contiene un AndroidManifest.xml como el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="org.example.hello"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk android:minSdkVersion="8" />

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:label="@string/app_name"
         android:name=".Hola" >
         <intent-filter >
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

Otra vista diferente del manifiesto es la siguiente:

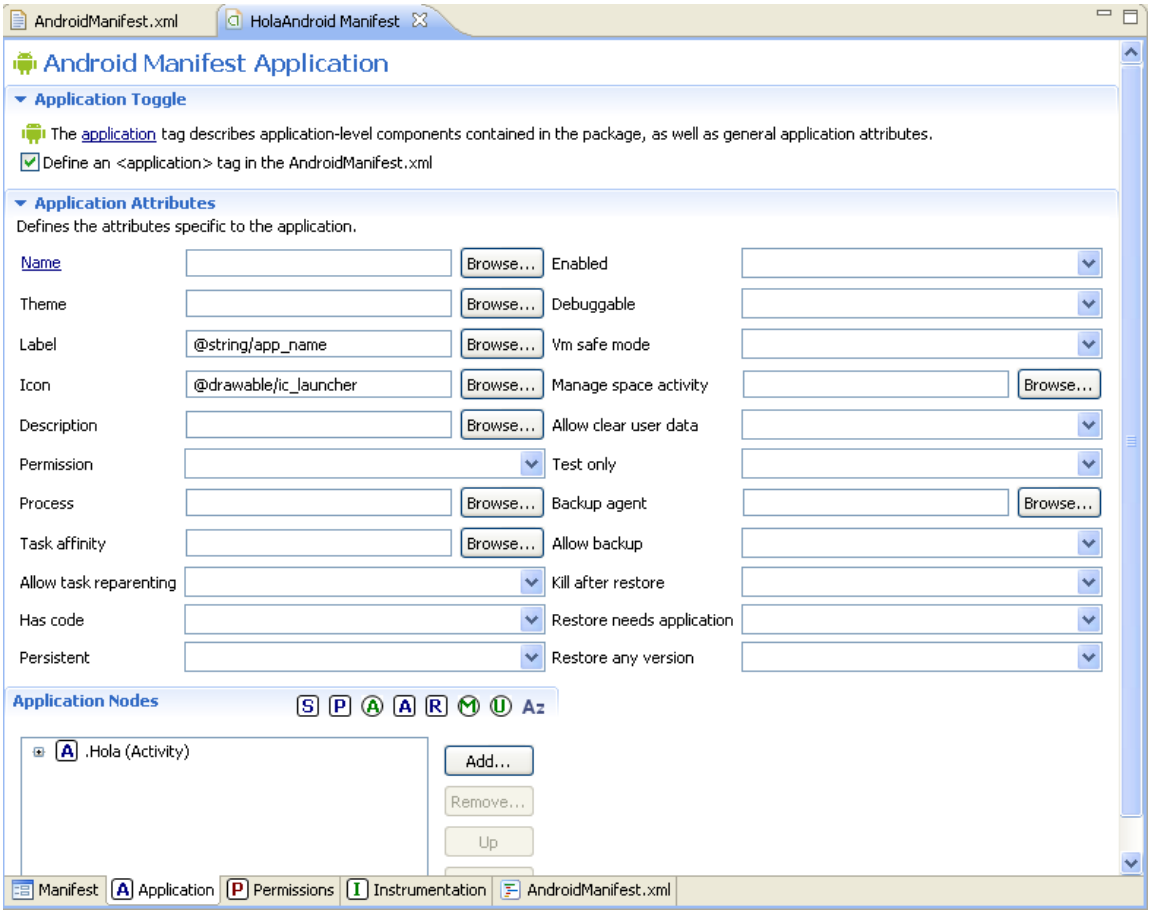


Figura 10. Editor Android Manifest

Y por fin, una vez explicadas cada una de las partes que componen el proyecto, vamos a crear nuestro primer proyecto con Android. Primero pinchar en "File"->"New"->"Other"->"Android Project", saldrá la pantalla que se muestra a continuación ("Create Android Project"). Simplemente en "Project Name" poner: HelloAndroid y pulsar "Next". En la siguiente pantalla, "Select Build Target", seleccionar la versión de Android sobre la que construir el proyecto. Vamos a seleccionar Android 2.2, para que nuestra aplicación pueda correr en cualquier terminal que tenga ésta versión o una posterior.

En la última pantalla antes de dar por concluida la configuración del nuevo proyecto, "Application Info", completar los siguientes campos:

- "Application Name": Hello, Android
- "Package Name": org.example.hello
- "Create Activity": Hello

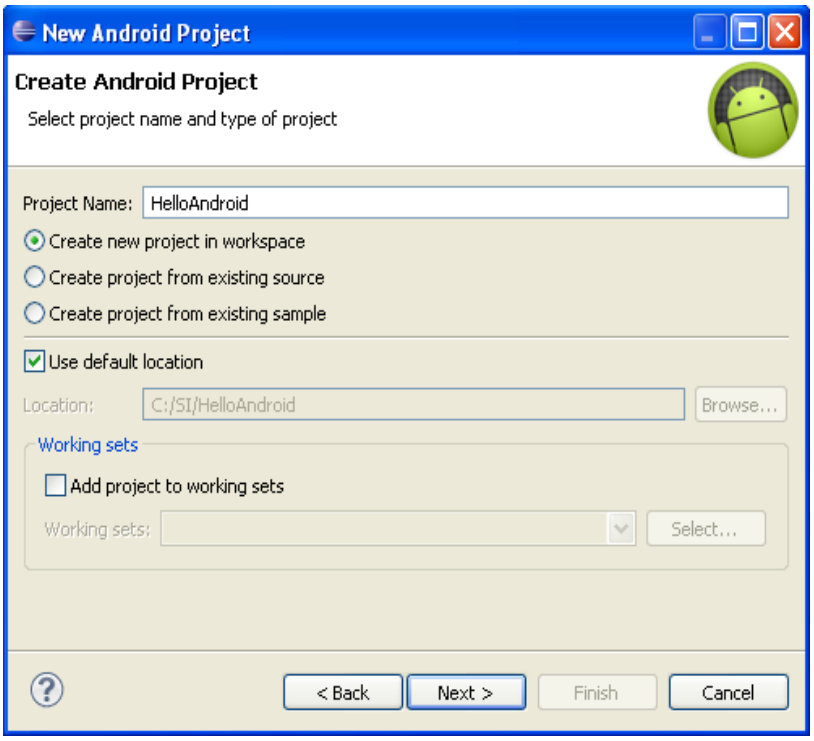


Figura 11. Create Android Project

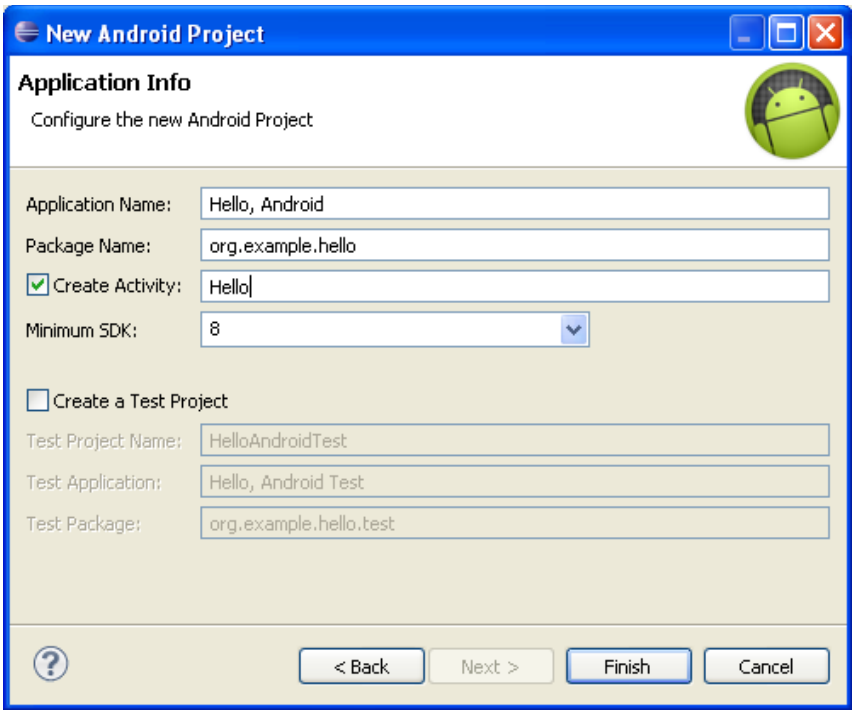


Figura 12. Application Info

Para ejecutar nuestra aplicación, primero debemos tener creado un emulador de nuestra versión de Android. Para ello, pinchar en el símbolo que abre el "Android Virtual Device Manager", y pulsar en "New":

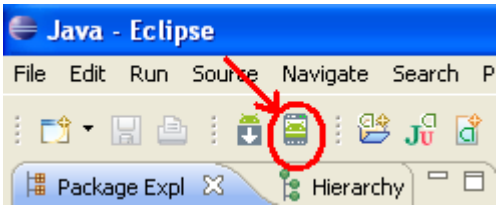


Figura 13. Símbolo Android Virtual Device Manager

En la siguiente pantalla rellenar los siguientes campos:

.- "Name": em2.2

.- "Target": Android 2.2 – API Level 8

.- "Size": 128 MiB

.- "Built-in": Default(WVGA800)

.- Si se quiere añadir funcionalidades Hardware, en "Hardware" pulsar "New", y seleccionar la

la opción/es deseada/s. En el ejemplo se ha añadido la funcionalidad "Camera support"

.- Por último, pulsar "Create AVD".

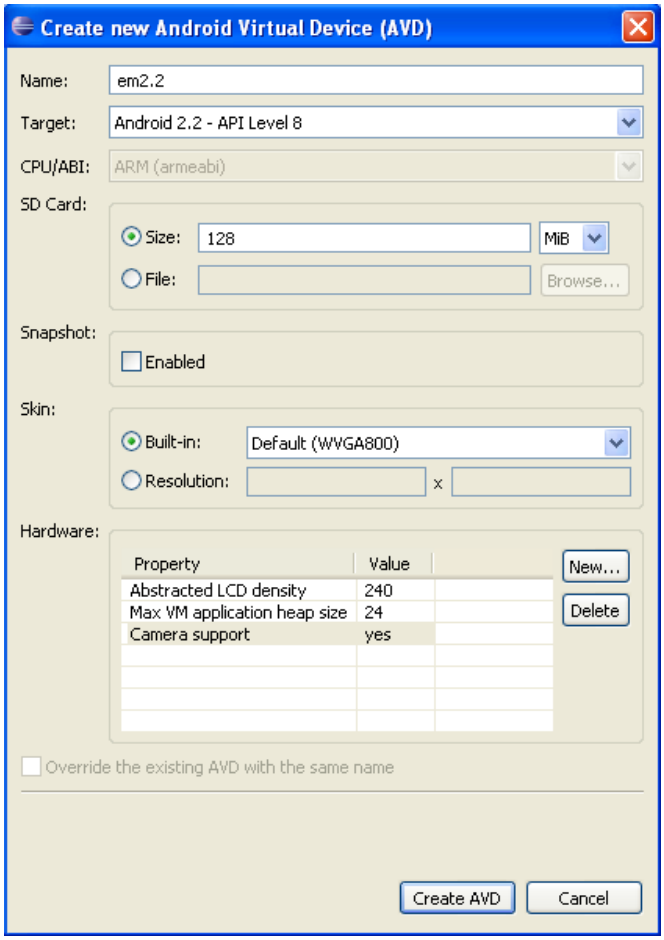


Figura 14. Create AVD

Ahora ya podemos ejecutar nuestra aplicación. Pinchar con el botón derecho del ratón sobre el proyecto, y en "Run As", seleccionar "Android Application". Se lanzará el emulador (hay que tener paciencia, pues debido a que consume muchos recursos, tardará un rato), y pasado un tiempo, se mostrará nuestro primer programa en Android:

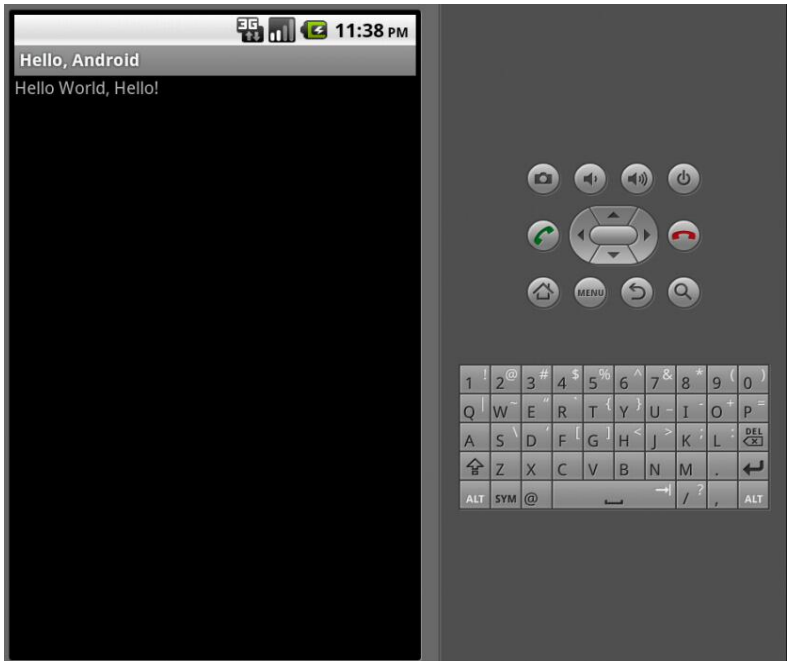


Figura 15. Simulación Hello Android

2. CONCEPTOS BÁSICOS

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

COMPONENTES DE UNA APLICACION

Para diseñar una aplicación en Android, es necesario tener claros los elementos que la componen y la funcionalidad de cada uno de ellos. Ya hemos visto el ejemplo del "Hola Android", por lo que podemos intuir algunos de ellos. Uno de los aspectos más importantes a tener en cuenta es su funcionamiento. Android trabaja en Linux, y cada aplicación utiliza un proceso propio. Se distinguen por el ID, un identificador para que solo ella tenga acceso a sus archivos. Los dispositivos tienen un único foco, la ejecución principal, que es la aplicación que está visible en la pantalla, pero puede tener varias aplicaciones en un segundo plano, cada una con su propia pila de tareas. La pila de tareas es la secuencia de ejecución de procesos en Android. Se componen de actividades que se van apilando según son invocadas, y solo pueden terminarse cuando las tareas que tiene encima están terminadas, o cuando el sistema las destruye porque necesita memoria, por lo que tienen que estar preparadas para terminar en cualquier momento. El sistema siempre eliminará la actividad que lleve más tiempo parada. En caso de que el sistema necesitase mucha memoria, si la aplicación no está en el foco, puede ser eliminada por completo a excepción de su actividad principal.

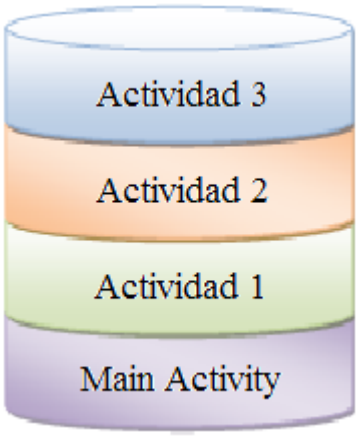


Figura 1. Pila de actividades Android

Una de las características principales del diseño en Android es la reutilización de componentes entre las aplicaciones, es decir, dos aplicaciones diferentes pueden utilizar una misma componente, aunque esté en otra aplicación para así, evitar la repetición innecesaria de código, y la consiguiente ocupación de espacio. Los componentes son los elementos básicos con los que se construyen el proyecto. Hay cuatro tipos, pero las aplicaciones se componen principalmente de actividades. Habrá tantas actividades como ventanas distintas tenga la aplicación. Sin embargo, por si solos, los componentes no pueden hacer funcionar una aplicación. Para ello están los *intents*.

Todos ellos deben declararse en el AndroidManifest.xml (junto con otros elementos que se mostrarán después) con el mismo nombre que lleve la clase asociada. Por ejemplo, la clase MainActivity, será definida en el AndroidManifest con el mismo nombre.

ACTIVIDADES

Una actividad (o Activity) es la componente principal encargada de mostrar al usuario la interfaz gráfica, es decir, una actividad sería el equivalente a una ventana, y es el medio de comunicación entre la aplicación y el usuario. Se define una actividad por cada interfaz del proyecto. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado (que se guarda en ./res/layout) para poder ser tratados en la clase NameActivity.class, que hereda de la clase Activity.

Dentro del fichero xml asociado a la actividad, se definen los elementos tales como ubicación de los elementos en la pantalla (layouts), botones, textos, checkbox, etc., cono se verá en capítulos posteriores. Las actividades tienen un ciclo de vida, es decir, pasan por diferentes estados desde que se inician hasta que se destruyen. Sus 3 posibles estados son:

- Activo: ocurre cuando la actividad está en ejecución, es decir, es la tarea principal
- Pausado: la actividad se encuentra semi-suspendida, es decir, aun se está ejecutando y es visible, pero no es la tarea principal. Se debe guardar la información en este estado para prevenir una posible pérdida de datos en caso de que el sistema decida prescindir de ella para liberar memoria.
- Parado: la actividad está detenida, no es visible al usuario y el sistema puede liberar memoria. En caso de necesitarla de nuevo, será reiniciada desde el principio.

Una vez definido el ciclo de vida, hay que tener en cuenta qué métodos son importantes en cada uno de ellos. Aquí están los métodos más importantes de una actividad:

- OnCreate (Bundle savedInstanceState): es el método que crea la actividad. Recibe un parámetro de tipo Bundle, que contiene el estado anterior de la actividad, para preservar la información que hubiera, en caso de que hubiera sido suspendida, aunque también puede iniciarse con un null si la información anterior no es necesaria o no existe.
- OnRestart(): reinicia una actividad tras haber sido parada (si continúa en la pila de tareas). Se inicia desde cero.
- Onstart(): inmediatamente después de onCreate(Bundle savedInstanceState), o de onRestart() según corresponda. Muestra al usuario la actividad. Si ésta va a estar en un primer plano, el siguiente método debe ser onResume(). Si por el contrario se desarrolla por debajo, el método siguiente será onStop(). Es recomendable llamar al método onRestoreInstanceState() para asegurar la información
- OnResume(): establece el inicio de la interactividad entre el usuario y la aplicación. Solo se ejecuta cuando la actividad está en primer plano. Si necesita información previa, el método onRestoreInstanceState() aportará la situación en que estaba la

actividad al llamar al onResume(). También puede guardar el estado con onSaveInstanceState().

- OnPause(): se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra. Guarda la información, para poder restaurar cuando vuelva a estar activa en el método onSaveInstanceState(). Si la actividad vuelve a primer plano, el siguiente método será onResume(). En caso contrario, será onStop().
- OnStop(): la actividad pasa a un segundo plano por un largo período. Como ya se ha dicho, el sistema puede liberar el espacio que ocupa, en caso de necesidad, o si la actividad lleva parada mucho tiempo.
- OnDestroy(): es el método final de la vida de una actividad. Se llama cuando ésta ya no es necesaria, o cuando se ha llamado al método finish().

Además de estos métodos, cabe destacar dos más, que son de vital importancia:

- OnSavedInstanceState(): guarda el estado de una actividad. Es muy útil cuando se va a pausar una actividad para abrir otra.
- OnRestoreInstanceState(): restaura los datos guardados en onSavedInstanceState() al reiniciar una actividad.

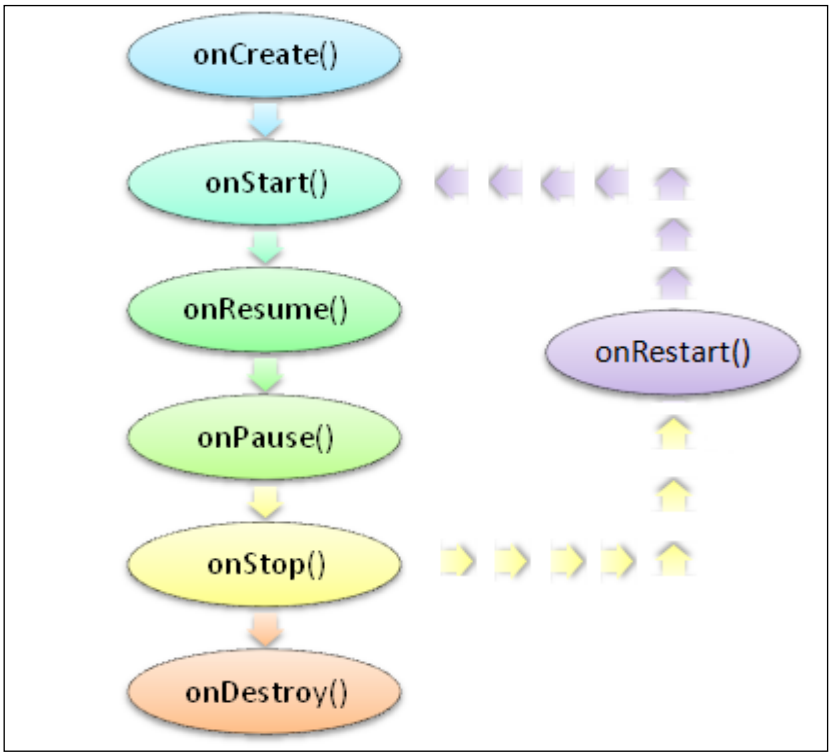


Figura 2. Ciclo de vida de una actividad

SERVICIOS

Los servicios (o service) son tareas no visibles que se ejecutan siempre por debajo, incluso cuando la actividad asociada no se encuentra en primer plano. Tiene un hilo propio (aunque no se pueden ejecutar solo), lo que permite llevar a cabo cualquier tarea, por pesada que sea. No necesita interfaz, a no ser que se pida explícitamente, en cuyo caso la clase Service la exportaría.

El ciclo de vida de un servicio se inicia con el método onCreate(Bundle), y se libera con el método onDestroy(). Sin embargo, el desarrollo puede llevarse a cabo de dos maneras, dependiendo de cómo se lance:

- Si se llama al método startService(), esto implicará que el servicio ejecutará todo su ciclo vital. El siguiente método tras onCreate(Bundle) será onStartComand(Intent, int, int). Para terminar el servicio externamente, se usa stopService(), e internamente, stopSelf() ó stopSelfResult(), ambos de la clase Service.
- En otro caso, si el servicio se llama con bindService(), el usuario podrá interactuar mediante la interfaz que exporta el servicio, y tras onCreate(Bundle) se ejecutará el método onBind(Intent). En este caso, el servicio se termina llamando al método onUnbind(Intent). También es posible reiniciarlo con el método onRebind(Intent).

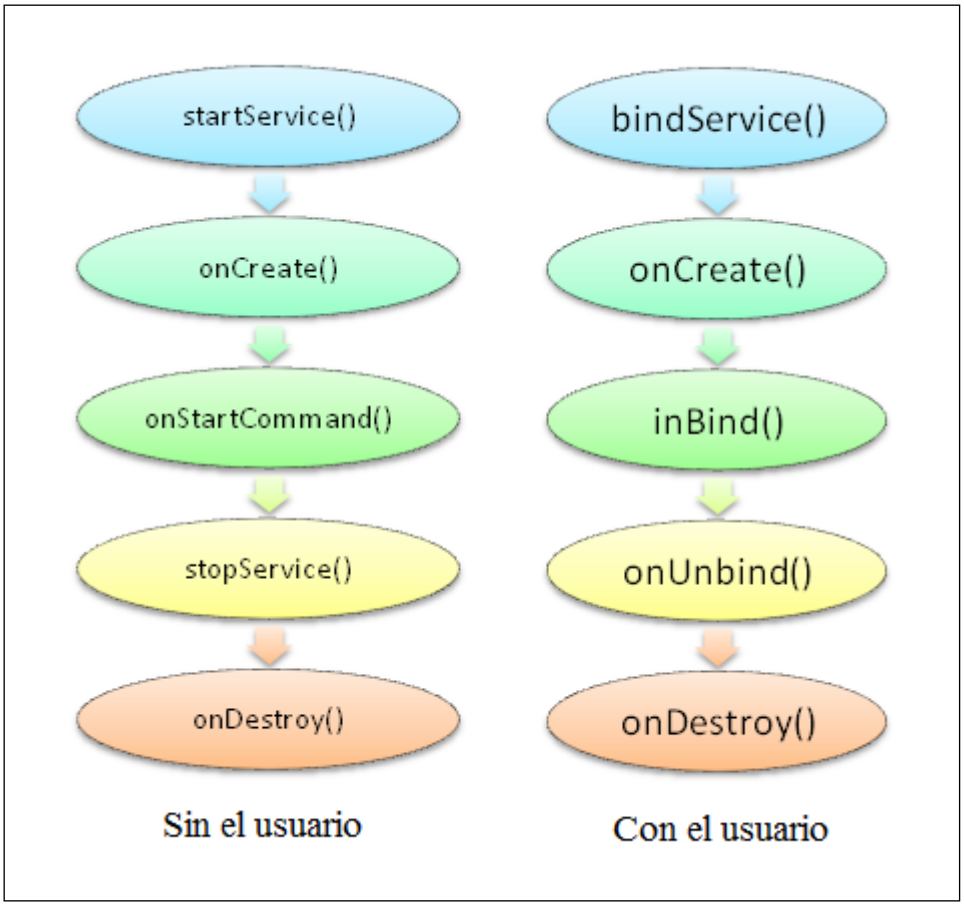


Figura 3. Ciclo de vida de un servicio

Receptores de Mensajes de Distribución

También llamados broadcast receiver o notificaciones, son los encargados de reaccionar ante los eventos ocurridos en el dispositivo, ya sean generados por el sistema o por una aplicación externa. No tienen interfaz, pero pueden lanzar una activity por medio de un evento. La clase que defina estos componentes heredará de la clase BroadCastReceiver. Su ciclo de vida es muy corto, ya que solo están activos mientras se ejecuta el método onReceive (Context, Intent), que es equivalente al onCreate(Bundle) de otros componentes. El objeto Context nos pasa es estado actual, y el intent, nos permitirá lanzar el evento.

Proveedores de contenidos

Estos proveedores en inglés llamados content provider, se encargan de que la aplicación pueda acceder a la información que necesita, siempre que se haya declarado el correspondiente provider en el AndroidManifest , compartiendo información sin revelar estructura u orden interno. Implementan una interfaz, pero se comunica con ella a través de la clase ContentResolver. Cada vez que se usa un ContentResolver, se activa un ContentProvider. Para obtener los datos necesarios, es necesario conocer la URI (identificador) del dato, los campos que tiene, y los tipos de esos campos. Con esto ya podemos llamar al método ContentResolver.query().

Intents

Los intents son el medio de activación de los componentes (excepto los content provider, que se activan usando ContentResolver). Contiene los datos que describen la operación que desarrollará el componente a quien va dirigido. Se declaran en el AndroidManifets con la etiqueta <Intent>. Pueden ser explícitos o implícitos. Los implícitos no especifican el componente al que va destinado, mientras que el explícito, si. Según el componente, los intents se tratan de diferentes maneras:

- Activity: los intents se lanzan desde el método starActivity(Intent) ó startActivitForResult(Intent). La información se extrae con el método getIntent().

  Los intents tienen definidas algunas acciones para las activity, es decir, informan de la acción a realizar. Entre ellas, por ejemplo se encuentra ACTION_CALL que inicia una llamada.

- Service: para este tipo de componentes, los intents se pasan a los métodos startService(Intent) o bindService(Intent) dependiendo del tipo de ciclo que escojamos. La información será extraída por el método getIntent() en el primer caso y onBind() en el segundo.

  Otra posibilidad es que el servicio sea lanzado por un intent, si aun no esta en funcionamiento.

- Broadcast Receiver: en este caso, el intent será enviado a todos los métodos que pueden recibir el intent : sendBroadcast(), sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle),

sendStickyBroadcast()…, que lo analizarán en su método onReceive(Context, Intent).

También tienen acciones definidas para este componente, aunque en este caso lo que hacen es informar de que ha ocurrido el evento. Por ejemplo tenemos ACTION_BATTERY_LOW, que informa de que la batería esta baja, o ACTION_SCREEN_ON, para cuando la pantalla se ilumina.

Intent-filters

Utilizados únicamente por los intents implícitos, los intent-filters definen (y delimitan) qué tipos de intent puede lanzar la actividad, o qué tipos de intent puede recibir un broadcast. Por ejemplo, para un intent que no especifica a que actividad va dirigido, se consulta el intent filter de una de ellas, y si lo satisface, el intent usará lanzará esa actividad. Se definen en el AndroidManifest con la etiqueta <intent-filter>. La información que pasan los intents debe estar contenida en la definición del intent filter para que la componente pueda ser activada (o pueda recibirlo en el caso del broadcast). Esta información se compone de tres campos:

- Action: string que informa del tipo de acción llevada a cabo. Las acciones pueden ser dadas por la clase Intent, por una API de Android o definidas por el diseñador.
- Data: informa del identificador (URI) del dato que se asocia a la acción y del tipo de ese dato. Es importante la coherencia ya que si la acción requiere un dato de tipo texto, un intent con un dato de tipo imagen no podría ser lanzado.
- Category: string que contiene información adicional sobre el tipo de componente al que va dirigido el intent. La lista de categorías esta incluida en la clase Intent

AndroidManifest

Como ya se introdujo en el tema anterior, este fichero es un documento xml en el que se declaran los elementos de la aplicación, así como sus restricciones, permisos, procesos, acceso a datos e interacciones con elementos de otras aplicaciones. Cada elemento se declara con una etiqueta única. No debe confundirse este documento con el xml asociado a cada actividad. Los elementos gráficos y distribución de la pantalla serán definidos para cada actividad dentro de su xml, pero no en el AndroidManifest. Al implementar el AndroidManifest se deben seguir unas pautas para hacer más comprensible el documento:

Código Android

```
<?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.rec.app"
        android:versionCode="1"
        android:versionName="1.0" >
```

```
<uses-sdk android:minSdkVersion="8" />
<application
        //Aquí se añadirán las acitivdades y sus permisos, acciones, etc…
</application>
```

Figura 4. Código generado automáticamente al crear el AndroidManifest

Este código es generado por el SDK a partir de la información que se ha proporcionado al crear el proyecto. Se declara el manifiesto con la etiqueta <*manifest* > y dentro se incluye el paquete en que se encuentra la aplicación y la versión del código. También incluye la versión del sdk que usa (con la etiqueta <uses-sdk>). A continuación, el usuario definirá la aplicación, incluyendo todos sus componentes en la etiqueta <application>. La declaración de componentes puede ser desordenada, pero para un mejor manejo de este fichero, se recomienda seguir algún tipo de orden. Las activities se declaran con la etiqueta <activity>. En ellas, lo primero es añadir el nombre de la actividad (android:name), que coincidirá con el de la clase en que se define el comportamiento. Además se pueden añadir imágenes, así como cambiar los atributos de que se dispone. A continuación, se declararían los intent filters asociados a la actividad, en caso de que los haya.

Los service se declaran con la etiqueta <Service> y aunque tienen menos atributos que las actividades, lo principal es darles un nombre y especificar si el sistema puede o no utilizarlo mediante el atributo enabled (android:enabled). Después irían los intent filters. Los broadcast receiver utilizan <receiver> y al igual que service, necesita los atributos name y enabled, así como intent filter en caso de necesitarlos. Todos los componentes anteriores declaran del mismo modo sus intent filters. Los content provider utilizan la etiqueta <provider> y son los únicos componentes en los que no se declaran intent filters, ya que no son necesarios. De nuevo el único atributo necesario es el nombre.

Un ejemplo práctico

A continuación se describe un ejemplo en Android aplicando los recursos aprendidos anteriormente. Al comienzo de un proyecto, tras crearlo en eclipse, lo único que tenemos es la MainActivity, o clase principal. En la siguiente imagen se muestra un ejemplo en el que se puede ver la estructura básica de una activity. Se sobrescribe el método onCreate(Bundle) y se muestra con el método setContentView( ). En la actividad se definirán las acciones propias de los elementos del xml. En este caso, tenemos un botón que, al presionar, abre una nueva actividad llamada ProductoActivity.

Código Android

```
package example.app


import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
        /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button boton = (Button) findViewById(R.id.botonInicial);
        boton.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                    Intent i = new Intent();
                    i.setClass(getApplicationContext(), ProductoActivity.class);
                    startActivity(i);
            }
        });
    }
```

Figura 5. Ejemplo de Activity

Es conveniente implementar la acción del botón back, pausando la actividad, sin destruirla. Además, también se puede incluir la típica pregunta "¿Está seguro de que desea salir?":

Código Android

```
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        switch (keyCode){
            case KeyEvent.KEYCODE_BACK:
            builder.setMessage("¿Seguro que deseas salir?");
```

```
            builder.setCancelable(false);
            builder.setPositiveButton("Sí", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    onPause();
                   System.exit(RESULT_OK);
                }   });
              builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                            //Acciones que se ejecutan al presionar el botón No
                }
            });
            break;
        }
        AlertDialog alert = builder.create();
        alert.show();
        return true;
    }
```

Figura 6: Implementación del botón back.

El archivo xml asociado a la MainActivity contendría el botón con el id botonInicial.

Código xml

```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="@drawable/prod_background"
        android:gravity="center"
        android:orientation="vertical" >
        <Button
            android:id="@+id/botonInicial"
            android:layout_width="112dp"
```

```
            android:layout_height="146dp"
            android:layout_gravity="center"
            android:background="@drawable/logo"
            android:gravity="center" />
</LinearLayout>
```

Figura 7. Ejemplo xml de Main activity

Una vez la actividad está definida, hay que tener claro qué recursos le serán necesarios. En este caso, y para que sirva de ejemplo, la actividad activará un Service (adecuadamente añadida en el androidManifest) que no tendrá que interactuar con el usuario, por lo que no habrá que sobrescribir el método onBind(Intent). Los métodos que se necesitan son onCreate(), startService(), onStartCommand y onDestroy(). En este ejemplo, el servicio reproducirá un sonido al inicio de la aplicación. De la gestión del sonido se ocupa la clase Reproductor.

Código Android

```
package example.app


import android.app.Service;
import android.content.Intent;
import android.os.IBinder;


public class MyService extends Service {
    private static Reproductor sonido;
    @Override
    public void onCreate() {
        sonido = Reproductor.create(this, R.raw.sonido_inicio);
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        sonido.start();
        onDestroy();
        return Service.START_NOT_STICKY;
    }
    @Override
```

```
    public IBinder onBind(Intent arg0) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Figura 8: Implementación de un servicio.

Cuando se ha creado el servicio, se añade en la actividad en la que será llamado, en este caso, la actividad Main:

Código Android

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
     @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button boton = (Button) findViewById(R.id.botonInicial);
        Intent servicio = new Intent(this, MyService.class);
        startService(servicio);
        boton.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                Intent i = new Intent();
                i.setClass(getApplicationContext(), ProductoActivity.class);
                startActivity(i);
            }
        });
}
```

Figura 9. Uso de Service en la clase MainActivity

Otro componente que se puede incluir en la aplicación es el broadcast receiver, por ejemplo para que al registrar una llamada entrante, la aplicación actúe de un modo determinado: se cierre, se suspenda o guarde la información existente.

Código Android

```
package com.rec.app;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.telephony.TelephonyManager;


public class MyBroadcastReceiver extends BroadcastReceiver {

        @Override

        public void onReceive(Context context, Intent intent) {

            String llamando =
   intent.getStringExtra(TelephonyManager.EXTRA_STATE_RINGING);

             if (!llamando.equals(TelephonyManager.EXTRA_STATE_RINGING)){

                 //Acciones llevadas a cabo por la aplicación cuando se recibe una llamada

            }

        }

   }
```

Figura 10. Ejemplo de Broadcast Receiver

Se puede añadir tantos componentes como sean necesarios. También es conveniente incluir algún proveedor de contenidos, pero para ello es necesario estudiar previamente el manejo de bases de datos. No se debe olvidar incluir todos estos componentes en el AndroidManifest.

Código xml

```
<?xml version="1.0" encoding="utf-8"?>

    <manifest xmlns:android="http://schemas.android.com/apk/res/android"

        package="com.rec.app"

        android:versionCode="1"

        android:versionName="1.0" >
```

```
    <uses-sdk android:minSdkVersion="8" />



<application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".MainActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".LocalizaActivity" />
        <activity android:name=".ProductoActivity" />

        <service android:name=".MyService" />

        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE" />
            </intent-filter>
        </receiver>
        <uses-permission android:name =
                "android.permission.READ_PHONE_STATE"/>
    </application>
</manifest>
```

Figura 11. Ejemplo AndroidManifest

En resumen, las aplicaciones en Android tienen diverso grado de dificultad, dependiendo de su funcionalidad, pero la estructura siempre es la misma. Uniendo estos conocimientos al uso de

recursos gráficos, bases de datos, mapas, y otros elementos, las posibilidades de estas aplicaciones son bastante amplias.

3. INTERFAZ DEL USUARIO

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

BREVE INTRODUCCIÓN

La interfaz de usuario es la principal sección de interacción entre persona y dispositivo. A todas las funcionalidades disponibles en una aplicación se accede a través de la pantalla, que es por donde se muestra. Es muy importante conseguir que el manejo sea intuitivo y sencillo, y que el aspecto visual sea atractivo.

Para construirla, se emplean diferentes objetos que veremos a continuación, todos ellos descendientes de la clase View. Fundamentalmente hay 2: los propios objetos de tipo View, como por ejemplo botones o etiquetas, y que son la base de una subclase llamada widgets; y los de tipo ViewGroup, que es una clase que extiende a View, y que son la base de una subclase llamada layouts.

La Figura 1 muestra un esquema de la estructura de la interfaz, por un lado están los nodos ViewGroup que son contenedores de elementos de tipo View e incluso de otros elementos de su mismo tipo. Por otro lado están los nodos tipo View, siempre hojas del árbol, que son los controles con los que el usuario interactúa.

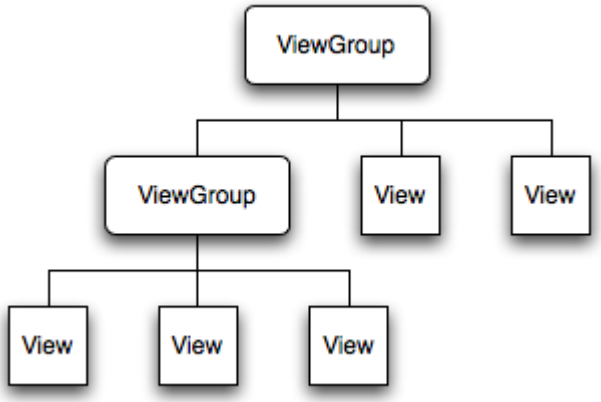


Figura 1. Estructura de la interfaz

LAYOUTS EN XML

Un layout es un recurso con el que puedes describir lo que quieres mostrar por pantalla y cómo lo quieres mostrar. La manera más común de crearlo es a través de un archivo XML (en el directorio res/layout del proyecto), con un formato muy similar a HTML, que tiene en siguiente patrón: <Nombre_del_layout atributo1=”valor1”… atributoN=”valorN”> elementos/componentes </Nombre_del_layout>. Una vez se ha creado el archivo XML que define el layout, hay que cargarlo desde el código de la aplicación, en el onCreate() de la actividad como parámetro del método setContentView():

```
public void onCreate(Bundle savedInstance) {
    super.onCreate(savedInstance);
    setContentView(R.layout.nombre_del_layout);
}
```

Dentro de los parámetros que tiene cada layouts, hay dos que son comunes a todos: layout_height que sirven para especificar el valor de la altura del entorno y layout_width que específica la anchura. Aunque se pueden emplear valores numéricos, si se pone “wrap_content”, el tamaño se ajusta a las dimensiones del contenido, y con “fill_parent”, será tan grande como lo permita su padre o contenedor. Estas 2 opciones son más recomendables que el ajuste manual. También se pueden establecer y consultar los márgenes, bordes y la posición del layout, entre otras opciones, mediante métodos y funciones a los que, en esta ocasión, se llaman desde el código de la aplicación.

Cada componente tiene su propia variedad de atributos en XML. El atributo “ID” se encarga de distinguirlos del resto, otorgándole un nombre único. Una vez establecido (mediante, por ejemplo, android:id="@+id/nombre"), para referenciarlo desde el código de la aplicación es preciso hacerlo de esta manera:

```
Tipo myTipo = (Tipo) findViewById(R.id.nombre);
```

Existen otros muchos atributos, que varían dependiendo del componente con el que estemos tratando, y con los que podemos establecer el color de fondo, tamaño, gravedad y un sinfín de opciones más.

Los principales tipos de Layout son:

FRAME LAYOUT

Es el más simple de todos los existentes. Todos los objetos que se introduzcan se situarán en la esquina superior izquierda, por lo que si hay más de uno, se ocultarán total o parcialmente entre ellos, salvo que los declaremos como transparentes. Por este motivo, su uso ideal es el de mostrar una sola imagen que complete toda la pantalla, como en la Figura 2.

Figura 2. Frame Layout

LINEAR LAYOUT

Se trata del Layout que viene por defecto, y uno de los más sencillos. Los objetos son estructurados horizontal o verticalmente, dependiendo del atributo "orientation" de su archivo XML correspondiente, y siempre en una única fila o columna, con un comportamiento similar al de una pila. A continuación se muestra un ejemplo del código de este tipo de layout:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">

<EditText android:id="@+id/NombreEditText"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />

<Button android:id="@+id/NombreButton"
android:layout_width="wrap_content"
android:layout_height="fill_parent" />
</LinearLayout>
```

TABLELAYOUT

Utilizando esta opción, se consigue una distribución tabular de los elementos de nuestra interfaz. El comportamiento es similar al empleado en HTML: se definen las filas, y dentro de ellas, las columnas. La tabla tendrá tantas columnas como la fila con un mayor número de celdas. En cada casilla, se podrá introducir el objeto deseado (e incluso dejarla vacía). También existe la posibilidad de combinar celdas. Por lo general, la dimensión de cada casilla la determina el elemento contenido en ella. No obstante, este comportamiento puede variarse utilizando diversos atributos. La Figura 3 muestra un ejemplo de este tipo de layout.

Figura 3. Table Layout

RELATIVE LAYOUT

Es la opción que ofrece más posibilidades, y por tanto, la más compleja. Básicamente, empleando este Layout podremos colocar cada elemento en el lugar que deseemos, basándonos en su posición relativa al padre (contenedor) o a otros elementos existentes, pudiendo modificar las distancias entre objetos al antojo del programador. La Figura 4 muestra un ejemplo de este tipo de layout.

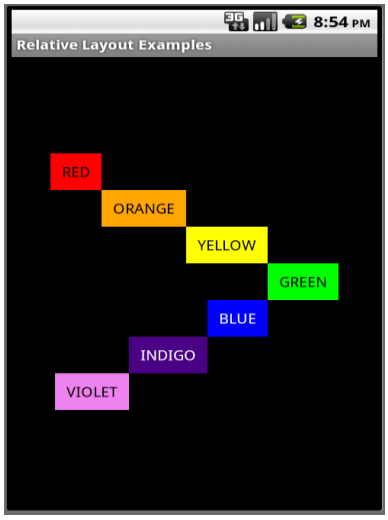


Figura 4. Relative Layout

EVENTOS DE USUARIO

Los eventos de usuario sirven para capturar la interacción del usuario con una determinada aplicación. Existen varias maneras de realizarlo.

EVENT LISTENERS

Un Event Listeners es una interfaz de la clase View a través de la cual se pueden detectar diferentes tipos de pulsación del usuario sobre el dispositivo. Los métodos más comunes que incluye esta interfaz son:

- **onClick()**: este método es llamado cuando el usuario hace una pulsación simple, ya sea por contacto, con teclas de navegación o de cualquier manera posible, sobre un determinado elemento de la interfaz. Una posible implementación, sobre un botón es la siguiente:

```
private OnClickListener nombreEvento = new OnClickListener() {
   public void onClick(View v) {
     // hacer lo que se desee
   }
};


Button boton = (Button)findViewById(R.id.corky);
boton.setOnClickListener(nombreEvento);
```

- **onKey():** este método es llamado si el usuario presiona determinada tecla o botón de su dispositivo mientras el cursor está situado sobre el elemento deseado.
- **onTouch():** este método es llamado cuando el usuario toca la pantalla y realiza una presión, se despega o se mueve por ella.

Estos métodos y otros existentes, hay que implementarlos en la Activity correspondiente o definirlos como una nueva clase anónima. Tras esto, deberemos pasar una instancia de la implementación a su respectivo método, con, por ejemplo, setOnClickListener() de OnClickListener.

EVENT HANDLERS

Si se está creando un componente de la clase View sobre cualquier tipo de Layout, se pueden definir varios métodos por defecto, denominados Event Handlers. Estos métodos son invocados cuando se produce un evento de cualquier tipo, como puede ser una pulsación sobre la pantalla.

TOUCH MODE

Determinados dispositivos pueden ser controlados a través de un cursor con sus respectivas teclas de movimiento, tocándoles la pantalla, o ambas. En el primer y último caso, es preciso contar con un selector de componente cuya función sea advertir sobre el elemento con el que se está tratando en cada momento, dependiendo del movimiento y situación del cursor. Sin embargo, en el momento que el usuario "toca" la pantalla, no es necesario resaltar el elemento con el que se va a trabajar. Por ello, en caso de tratar con un dispositivo que permita ambos tipos de navegación a través de su interfaz, existirá un modo llamado "touch mode", que determinará si es necesario resaltar el componente con el que se va a trabajar o no.

HANDLING FOCUS

Por lo comentado en el punto anterior, el sistema debe encargarse del selector de componente, manejando una rutina, en respuesta a la entrada dada por el usuario. Esto incluye el dónde situar el foco si un elemento es suprimido u ocultado, basándose en un algoritmo que selecciona al componente vecino más cercano.

MENÚS Y BARRAS DE ACCIONES

MENUS

Los menús son la forma más habitual de proporcionar al usuario una serie de acciones a realizar, ya sea sobre una aplicación o sobre las opciones del propio dispositivo. Para crear un menu la mejor opción es definirlo en un archivo XML que irá contenido en el directorio res/menú del proyecto. Los elementos que componen este fichero son: <menú>, que es el contenedor principal del resto de elementos; <ítem>, que representa cada una de las opciones que ofrece el menú, y <group>, que posibilita agrupar los "ítems" del menú a gusto del usuario, y que puede ser visible o no. A continuación se muestra un ejemplo del código de un menu:

```
<?xml version="1.0" encoding="utf-8"?>
<menu  xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/Opcion1" android:title="Opción1"
     android:icon="@drawable/miIcono1"></item>
<item android:id="@+id/ Opcion2" android:title="Opción2"
     android:icon="@drawable/ miIcono2"></item>
</menu>
```

Hay 3 tipos de menús de aplicación:

- **Menú principal:** es la colección primaria de una actividad, que aparece cuando el usuario acciona el botón "Menú" del dispositivo (ver Figura 5). Una forma de crearlo es la siguiente:

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.miMenu, menu);
    return true;
}
```

Figura 5. Menú principal

- **Menú contextual:** es una "lista" que surge en la pantalla que aparece cuando el usuario mantiene presionado un elemento determinado(ver Figura 6). Se implementa como el caso anterior, pero añadiendo un nuevo elemento de tipo <menu> al mismo nivel que los elementos <ítem>.

Figura 6. Menú contextual

- **Submenús:** es una lista de opciones que surge cuando el usuario acciona un elemento del menú, que contiene otro menú anidado.

BARRAS DE ACCIÓN

Las barras de acciones son barras de título que, además de mostrar algún tipo de información, pueden ejecutar acciones. El aspecto de ésta es acorde con el de la aplicación a la que pertenece. Es importante saber que sólo están disponibles para versiones de Android a partir de la 3.0 (versión 11 del sdk).

DIÁLOGOS Y NOTIFICACIONES

Los diálogos y las notificaciones son avisos o comprobaciones que surgen de una determinada aplicación, en forma de ventana.

DIALOGOS

La principal diferencia entre los dialogos y las notificaciones es que las segundas son meramente informativas, no se ejecutan en primer plano, y no requieren interacción directa con el usuario, mientras que los primeros sí se realizan en primer plano y pueden necesitar esa interacción para llevar a cabo una tarea, aunque se puede decir que un diálogo es un tipo de notificación. Un diálogo se crea con el método "onCreateDialog(numero)" dentor de una actividad, y se muestra como tal, con el método "showDialog(numero)", donde "numero", en ambos casos, representa un identificador único de cada dialogo. Para rechazarlos, hay que usar los métodos dismiss() o dismissDialog(int). Si se desea modificar el texto a mostrar o cualquier otro atributo, existe una colección de métodos, variable en función del tipo concreto del diálogo, que facilitan la labor.

Podemos enumerar 3 tipos de diálogos:

- **AlertDialog:** Se recomienda usarlo para mostrar un título, un mensaje de texto, uno o varios botones, o una lista de elementos para seleccionar. La Figura 7 muestra este tipo de diálogos.

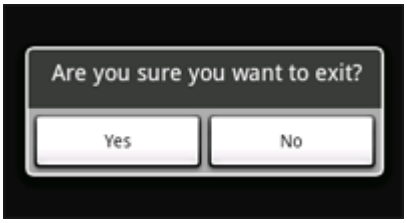


Figura 7. Diálogo de Alerta

Si el diálogo tiene botones es preciso incorporar un evento que detecte su pulsación. En el diálogo del de la Figura 7 el código sería el siguiente:

:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
              MyActivity.this.finish();
          }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
              dialog.cancel();
          }
       });
AlertDialog alert = builder.create();
```

En caso de que se desee incorporar una lista de elementos, es preciso crear un array de strings, donde cada elemento corresponderá a cada opción que se mostrará por pantalla, y posteriormente añadirla al diálogo de manera similar a como se ha visto para los botones.

- **ProgressDialog:** Su función es indicar que una tarea está siendo llevada a cabo. Si se conoce el esfuerzo que va a requerir dicha tarea, muestra una barra de progreso que informa sobre lo que se lleva completado y lo que resta; y si no se conoce, muestra un haz que describe un movimiento circular (ver Figura 8). Este tipo de diálogo puede incorporar botones, por ejemplo para cancelar el progreso.

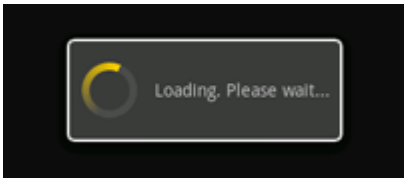


Figura 8. Diálogo de Progreso

- **Diálogo personalizado:** Android da la opción de crear un diálogo con los elementos/aspecto que el usuario desee. Para ello, hay que crear un layout en XML, guardarlo como "custom_dialog.xml", y añadirlo como vista al crear el diálogo.

NOTIFICACIONES

Hay 2 tipos de notificaciones:

- **Toast Notification:** Este tipo de notificación aparece sobre la ventana en la que estemos trabajando, y solamente ocupa el espacio que necesite el mensaje que muestra (ver Figura 9). No cierra ni modifica la actividad que se esté llevando a cabo, ni acepta interacción con el usuario, por lo que se desvanecerá en un periodo corto de tiempo.

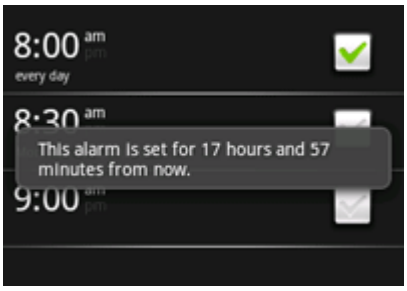


Figura 9. Toast Notification

Para crearla, mostrarla y situarla, es suficiente con incorporar unas instrucciones similares a las mostradas a continuación:

```
Toast toast = Toast.makeText(context, text, duration);
toast.show();
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

También se puede crear un layout (en este manual ya se ha visto cómo) y utilizarlo como una notificación de este tipo.

- **Status Bar Notification:** Esta notificación añade un mensaje en la ventana de notificaciones del sistema, y un icono en la barra de estado que, al seleccionarlo, lanza la aplicación que ha activado la notificación. Además, se pueden configurar de tal modo que al producirse una notificación de este tipo, el dispositivo suene, vibre o alerte al usuario de alguna manera. En la Figura 10 se puede ver un ejemplo de este tipo de notificación.

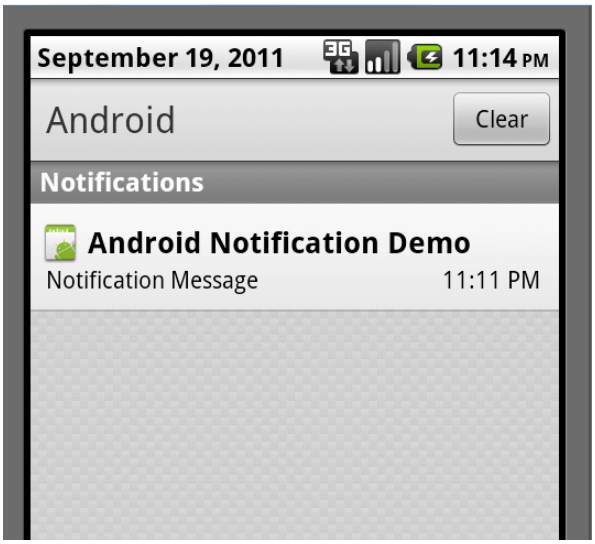


Figura 10. Status Bar Notification

Este tipo de notificaciones sólo deben referirse a actividades que se estén ejecutando en "background", y no podrán pasar a "foreground" sin la acción del usuario.

ESTILOS Y TEMAS

Un estilo es una colección de propiedades que permite especificar el aspecto y formato de una vista o una ventana. Un tema es lo mismo que un estilo, pero aplicado a una actividad al completo, no sólo a una vista. Android contiene una gran cantidad de estilos y temas predefinidos a completa disposición del usuario.

Para definir uno o varios estilos, hay que crear un archivo XML en el directorio res/values/ del proyecto. Todos los estilos que se quieran crear hay que definirlos dentro de ese documento, entre las etiquetas globales (<resource> </ resource >). Por cada uno de los estilos, hay que añadir un par de etiquetas <style></style>, y entre ellas hay que definir las propiedades del estilo. Por cada propiedad hay que añadir elemento <ítem>, identificándolo con un nombre. El siguiente código muestra como crear dos estilos llamados "NombreEstilo1", "NombreEstilo2".

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="NombreEstilo1"
parent="@android:style/TextAppearance.Medium">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#00FFFF</item>
      <item name="android:typeface">monospace</item>
   </style>

   <style name="NombreEstilo2"
parent="@android:style/TextAppearance.Medium">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:textColor">#FF0000</item>
   </style>
</resources>
```

Para aplicar un estilo a una vista individual, es suficiente con añadir el atributo "style" al elemento deseado dentro del layout, en el archivo XML correspondiente. Por ejemplo:

```
<TextView    style="@style/NombreEstilo1"   android:text="@string/hola" />
```

Para aplicar el estilo a una actividad (aplicación) al completo, es preciso incorporar el atributo "android:theme" en el <activity> (<application>) del Android Manifest. Por ejemplo:

```
<activity android:theme="@style/ NombreEstilo1">
```

4. RECURSOS DE APLICACIÓN

Luis Cruz – José Rodríguez de Llera – Alvaro Zapata

DEFINIENDO RECURSOS

Para la programación en Android, resulta conveniente separar los recursos que vaya a necesitar la aplicación (como imágenes u otro tipo de variables), de su código, de tal modo que se puedan mantener independientemente. Dentro de un proyecto de Android el directorio res/, contiene todos los grupos de recursos, y permite contener dentro otros tales como: animator/, anim/, color/, drawable/, layout/, menú/, raw/, values/, xml/. Excepto la 5ª y la 7ª opción de la lista anterior, que soportan recursos de tipo Bitmap o archivos arbitrarios, respectivamente, el resto sólo pueden contener documentos en XML.

También se debe especificar un recurso para diferentes configuraciones posibles de los dispositivos, puesto que mientras se ejecuta una aplicación, el sistema operativo Android, utiliza los recursos apropiados para la configuración activa. Es decir, conviene tener recursos alternativos (drawable, por ejemplo), por si el que tenemos por defecto no se ajusta apropiadamente cuando se utiliza un dispositivo no habitual.

Para especificar una configuración alternativa, hay que crear una nueva carpeta dentro de res/, del estilo <resources_name>-<config_qualifier>, que representan el directorio por defecto de un recurso determinado, y una configuración individual sobre en qué caso se puede usar ese recurso, respectivamente. Se permite añadir más de un <config_qualifier>, pero siempre separados con "-". Una vez creada la carpeta se incorparan en ella los recursos alternativos, que deben tener el mismo nombre que los que son "por defecto". A continuación se muestra un ejemplo (hdpi se refiere a dispositivos con densidad alta de pantalla):

```
res/
    drawable/
        icono.png
        fondo.png
    drawable-hdpi/
        icono.png
        fondo.png
```

Respecto a las configuraciones disponibles que pueden ajustarse dentro del ya mencionado <config_qualifier>, las posibilidades son amplísimas. La anchura, el idioma, el tamaño de la pantalla, la orientación o la densidad en píxeles, son sólo alguno de los campos existentes, y dentro de cada uno de ellos, hay diferentes opciones.

En caso de utilizar más de una configuración, es muy importante conocer las restricciones.

- El orden de los elementos, en el nombre, debe ser el correcto (siguiendo una tabla/lista que se puede consultar en la web de Android), puesto que de no ser así no serían tenidos en cuenta

- Tampoco se permiten anidar las carpetas. Todas deben ir en res/.
- Las mayúsculas no son tenidas en cuenta, puesto que se transforma todo a minúsculas al compilar.
- Por supuesto, no se puede incorporar más de un valor por cada tipo.

A continuación se muestran tres ejemplos no válidos:

```
drawable -hdpi-port / (orden)
res/ drawable /drawable-en/ (anidar)
drawable-rES-rFR/ (sólo un elemento por tipo)
```

La manera correcta de ponerla sería:

```
drawable-port-hdpi/
res/drawable-en/
drawable-rES /
```

Cuando existe un recurso que se quiere utilizar en más de una configuración (sin ser el "por defecto"), Android permite no tener que duplicarlo. Para ello, es preciso asignarle un nombre que lo diferencie del resto, e introducirlo en la carpeta por defecto. Esto se puede hacer con los tipos "layout", "drawable" y "strings y otros valores simples".

Android selecciona qué recurso utilizar durante la ejecución de una aplicación, obteniendo la configuración del dispositivo en que se realiza, y comparándola con los recursos disponibles en el proyecto. Para realizar esta búsqueda de recursos el sistema operativo realiza los siguientes pasos (la Figura 1 muestra un esquema de este algoritmo):

1. Eliminar las carpetas con recursos contradictorios a la configuración del dispositivo.
2. Coger el siguiente calificador con precedencia más alta, de la tabla que podemos encontrar en la Web de Android, mencionada anteriormente
3. ¿Está ese recurso en algún directorio?
   - Si no es así, volver al paso 2.
   - Si está, seguir con el paso 4.
4. Eliminar los directorios que no contengan ese calificador.
5. Repetir los pasos 2, 3 y 4 hasta que sólo quede 1 directorio.

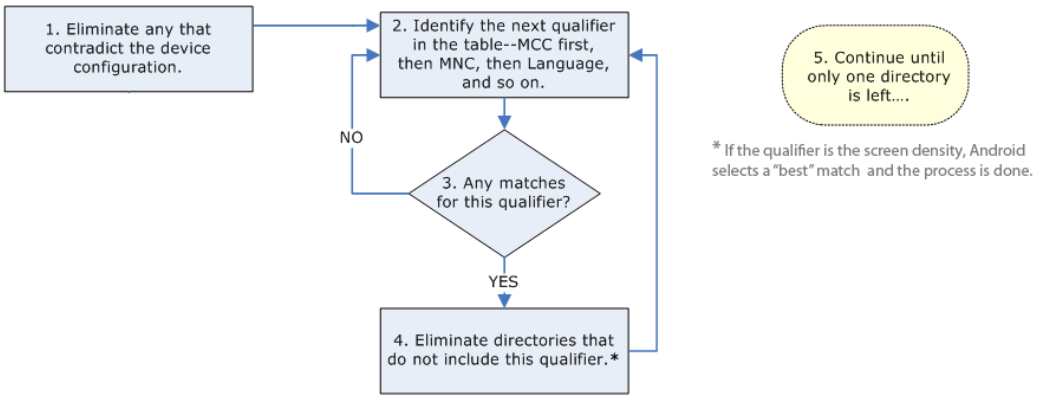


Figura 1. Algoritmo de búsqueda de recursos

Además de este algoritmo, el sistema, más adelante, optimiza algunos aspectos, siempre en función de la configuración del dispositivo.

UTILIZANDO LOS RECURSOS

Una vez que se han creado todos los recursos, se puede referir a ellos a través de su identificador. Todos ellos están definidos en la clase R del proyecto. Ésta se genera automáticamente al compilar, y contiene identificadores únicos, compuestos siempre por un tipo y un nombre, para cada recurso existente en el directorio res/. Hay 2 formas de acceder a ellos:

A TRAVÉS DEL CÓDIGO

A través del código, como una subclase de R. Se puede usar un recurso pasando su identificador como parámetro de una función, siguiendo el patrón:

```
[<package_name>.]R.<resource_type>.<resource_name>
```

Este patrón contiene el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), la subclase de R con el tipo y el nombre (sin su extensión) del recurso, respectivamente.

Esta forma de acceder a los recursos se puede utilizar para establecer una imagen de fondo, un layout para una pantalla o establecer el título de una actividad. Por ejemplo: getWindow().setBackgroundDrawableResource(R.drawable.fondo);

DESDE XML

En este caso, se usa la sintaxis correspondiente a la configuración del identificador del recurso. Se pueden definir elementos o atributos de un fichero XML como referencia a ellos. La sintaxis sigue el patrón:

```
@[<package_name>:]<resource_type>/<resource_name>
```

Donde "package_name" representa el paquete donde se encuentra el recurso (no es necesario poner nada si se referencia desde el propio paquete), "resource_type" es el tipo del recurso y "resource_name" el nombre del recurso sin su extensión.

Esta forma de acceder al recurso se puede utilizar para establecer el color de un texto o qué debe mostrar, o incluso para crear alias. A continuación se muestra un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<EditText
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/azul"
    android:text="@string/hola" />
```

También se puede establecer un atributo como referencia a un estilo perteneciente al tema que está en uso, incluyendo, por ejemplo,

```
android:textColor="?android:colorSecundario"
```

LOCALIZACIÓN

Android está a disposición del usuario en muchas regiones y dispositivos diferentes. Lo ideal, es que las aplicaciones utilicen los textos, la moneda, los números… de acuerdo a donde se estén ejecutando. Todo eso se controla incorporando la mayor parte del contenido de la interfaz de usuario a los directorios de recursos (como hemos visto anteriormente), y manejando el comportamiento de la interfaz desde el código Java. Siempre que una aplicación se ejecute de forma "local", y no haya definidos un texto específico para ella, Android cargará el "string.xml" por defecto. Sin embargo, si no existe ese archivo, se mostrará un error y la aplicación no se ejecutará.

La prioridad de los recursos en función de la localización es absoluta. Por ejemplo, si nos encontramos en Francia, y tenemos un dispositivo que soporta alta calidad de gráficos, dará prioridad al directorio res/drawable-fr/, independientemente de si en su interior los recursos definidos son para dispositivos que soportan exclusivamente gráficos de baja calidad, frente a un directorio denominado res/drawable-hdpi/, que inicialmente parecería más adecuado.

Se debe saber que los calificadores MCC y MCN son una excepción y son siempre prioritarios, por lo que los recursos contenidos en semejantes carpetas serán cargados por delante de cualquier otro. En ocasiones es preciso crear un layout flexible, de tal modo que se adecúe a un contexto, pero que no se limite a él. Por ejemplo, si el entorno local es España, la agenda de contactos es preciso que tenga campos para nombre y 2 apellidos, sin embargo, si el entorno es Irlanda, sobraría uno de los huecos para los apellidos. Para solucionar esta situación no es necesario duplicar los recursos, puesto que Android permite crear layouts que contienen campos que se activan o desactivan en función del idioma. Se puede utilizar Android para buscar recursos locales, a través de la siguiente instrucción:

```
String locale =
context.getResources().getConfiguration().locale.getDisplayName();
```

PROBAR APLICACIONES LOCALIZADAS

Para probar cómo funcionarían aplicaciones en otra localización, se puede cambiar la configuración del emulador con adb shell, siguiendo éstos pasos:

1. Elegir qué región se quiere simular y determinar su lenguaje y códigos de región, por ejemplo fr de francés y CA de Canadá.
2. Lanza el emulador.
3. Ejecuta mediante comandos, desde el ordenador que lanza el emulador, "abd Shell", o "adb –e Shell", si existe un dispositivo adjunto.
4. Una vez dentro, se debe ejecutar el siguiente comando reemplazando los corchetes por los códigos apropiados (fr y Ca respectivamente, para este ejemplo):
5. setprop persist.sys.language [language code];setprop persist.sys.country [country code];stop;sleep 5;start
6. reemplazando los corchetes por los códigos apropiados (fr y Ca respectivamente, para este ejemplo).

Estos pasos reiniciarán el emulador, y cuando se vuelva a ejecutar la aplicación, se hará sobre la nueva configuración. De hecho, cuando esté lista, se pueden publicar en el Market para diferentes localizaciones, ya sea con diferentes apk's, o incluyendo todos los recursos en la misma.

Una forma sencilla de comprobar si la aplicación tiene los recursos necesarios para ejecutarse, independientemente de la configuración, consiste en establecer el emulador en una localización para la que la aplicación no tenga recursos definidos. Si al ejecutarla nos sale un mensaje de error, es que faltan los recursos por defecto, o no están bien definidos.

TIPOS DE RECURSOS

En Android existen varios tipos de recursos, los más relevantes son:

MENU

Este recurso define al Menú Principal, Menú Contextual y Submenú. Hay que añadirlo en la carpeta res/menú/nombre_del_archivo.xml y se referencia mediante R.menu.nombre_del_archivo (en JAVA) y [package:]menú.nombre_del_fichero (en XML). A continuación se muestra el código de un menú con todas las opciones:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@[+][package:]id/resource_name"
          android:title="string"
          android:titleCondensed="string"
```

```
          android:icon="@[package:]drawable/drawable_resource_name"
          android:onClick="method name"
          android:showAsAction=["ifRoom" | "never" | "withText" |
"always" | "collapseActionView"]
          android:actionLayout="@[package:]layout/layout_resource_name
"
          android:actionViewClass="class name"
          android:actionProviderClass="class name"
          android:alphabeticShortcut="string"
          android:numericShortcut="string"
          android:checkable=["true" | "false"]
          android:visible=["true" | "false"]
          android:enabled=["true" | "false"]
          android:menuCategory=["container" | "system" | "secondary" |
"alternative"]
          android:orderInCategory="integer" />
    <group android:id="@[+][package:]id/resource name"
           android:checkableBehavior=["none" | "all" | "single"]
           android:visible=["true" | "false"]
           android:enabled=["true" | "false"]
           android:menuCategory=["container" | "system" | "secondary"
| "alternative"]
           android:orderInCategory="integer" >
        <item />
    </group>
    <item >
        <menu>
          <item />
        </menu>
    </item>
</menu>
```

Los elementos de este código son:

- <menu>: Requerido. Tiene que ser el nodo raíz e incluir el atributo xmlns:android obligatoriamente.
- <ítem>: Tiene que ser el hijo de un elemento de tipo <menu> o <grupo>. Si además contiene un elemento del 1er tipo, existirá un Submenú.
- <group>: Para crear una colección de elementos (de tipo <item>. Los contiene), que pueden compartir unas características. Es necesario que sea hijo de un elemento <menú>.

ANIMATION

Este recurso define animaciones predeterminadas. Las denominadas "Tween" se guardan en res/anim/ y son accedidas mediante la clase R.anim. Las "Frame" se almacenan en res/drawable/ y se accede a ellas desde R.drawable.

COLOR STATE LIST

Este recurso define colores que cambian basándose en la "View". Se guardan en res/color/ y se accede a ellos desde la clase R.color.

DRAWABLE:

Este recurso define diferentes tipos de gráficos con Bitmaps o con XML. Son accedidos a través de R.drawable y almacenados en res/drawable/.

LAYOUT:

Este recurso define la estructura general de la interfaz de usuario. Se guardan en res/layout/ y se accede a ellos desde la clase R.layout.

STRING:

Este recurso define strings, arrays de strings y plurales, incluyendo formato y estilo de ellos. Se almacenan en res/values/ y se pueden acceder desde las clases R.string, R.array y R.plurals respectivamente.

STYLE:

Este recurso define el aspecto y formato de los elementos de la interfaz de usuario. Se guardan en res/values/ y se accede a ellos desde la clase R.values.

OTROS:

Existen otros recursos que definen valores tales como booleanos, enteros, dimensiones, colores u otros arrays. Todos ellos se almacenan en res/values/, pero cada uno se accede desde subclases de R únicas, tales como R.bool, R.integer, R.dimen, etcétera.

Android permite definir recursos que se ajusten a diferentes configuraciones en diferentes dispositivos. A la hora de ejecutar una aplicación, sólo se utilizarán los adecuados y necesarios en función de cada situación. El resultado de ello es una optimización en la relación esfuerzo-calidad, para cada dispositivo.

5. DATOS EN ANDROID. IDEAS PRINCIPALES

Manuel Báez – Jorge Cordero – Miguel González

INTRODUCCIÓN

En Android hay cuatro maneras de almacenar los datos, se usa una u otra dependiendo de la función que tengan dichos datos. Es importante destacar que los datos de cada aplicación son privados a dicha aplicación, pero tenemos la posibilidad de compartirlos si así lo deseamos, como veremos más adelante.

En función de lo que queramos almacenar, los datos se dividen en cuatro tipos distintos:

- Preferencias: Conjunto de datos, clave/valor
- Archivos locales: Almacenamiento interno y externo (SD)
- Base de datos: SQLite
- Proveedores de contenidos: Único mecanismo para compartir datos, no se trata con ellos directamente.

PREFERENCIAS COMPARTIDAS

Este tipo de datos son una forma ágil para poder guardar datos simples de la aplicación. Estos datos son un conjunto clave/valor que perdura después de que la aplicación se cierre, por lo que es principalmente útil para guardar, como su nombre indica, las preferencias de una aplicación. Por ejemplo: guardar el idioma, configuraciones de sonido de la aplicación, fuentes, colores, etc. Los datos que permite guardar son primitivos y se invocan con los siguientes métodos putInt(), putBoolean(), putFloat(), putLong(), putString(), putStringSet() de la forma (String key, tipo value).

Para guardar los datos de las preferencias se utiliza una clase que hereda de PreferenceActivity, la cual guarda en un xml las preferencias de la aplicación que, en principio, serán cargadas cuando la iniciemos y guardadas cuando la cerremos. Dicho xml se guarda en /data/data/nombre_paquete/shared_prefs/preferencias_por_defecto.xml. Es importante destacar que podemos tener una colección de preferencias, para ello tendremos que asociarles un identificador.

A continuación se muestra un posible código del método onStop() que se usa para guardar los datos preferentes, también se puede usar el método onSaveInstance(). Ambos se invocan antes de cerrar la aplicación, siempre y cuando se cierre de manera correcta. . En dicho código se puede ver que se invoca al método edit(void) para editar las preferencias con SharedPreferences.Editor, se añaden valores con el métodos editor.putString() y se actualizan los nuevos valores usando commit(void)

Código Android

```
protected void onStop(){
    super.onStop();
     // Necesitamos un objeto Editor para poder modificar las preferencias
    SharedPreferences preferencias = getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = preferencias.edit();
    editor.putString("NumeroAlAzar", "58472");
    // Por ejemplo un número al azar
    // Actualizamos las nuevas preferencias
    editor.commit();
}
```

Una vez guardadas las preferencias hay que cargarlas, el siguiente código muestra cómo hacerlo:

Código Android

```
public class PreferenciasActivity extends PreferenceActivity implements
OnSharedPreferenceChangeListener {
//...     Tu código
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
              // Carga las preferencias del XML.
        addPreferencesFromResource(R.xml.preferencias);
getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
    }
//...     Tu código
}
```

Estas preferencias también hay que guardarlas en un fichero xml, como se muestra a continuación. Estos ficheros se almacenan en /res/xml.

Código xml

```
<PreferenceScreen  xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory>
            TU CÓDIGO DE ESTA CATEGORÍA
        </PreferenceCategory>
        <PreferenceCategory>
            TU CÓDIGO DE ESTA CATEGORÍA
        </PreferenceCategory>
</PreferenceScreen>
```

Una vez definidas las preferencias, hay que implementar una nueva actividad, para ello se crea una clase que extiende de PreferenceActivity de la siguiente forma:

Código Android

```
public class Opciones extends PreferenceActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.opciones);
   }
}
```

Hay que añadir esta nueva actividad al Manifest, para ello bastará con poner el siguiente código en el Manifest.

Código Android

```
<activity android:name=".PantallaOpciones"
   android:label="@string/app_name">
</activity>
```

ARCHIVOS LOCALES

Los datos, que no se ajusten al patrón clave/valor de las preferencias, hay que guardarlos como *archivos locales*. Dichos archivos, por defecto, no son accesibles desde otras aplicaciones y están regulados por permisos Unix. El acceso a estos datos es parecido al de Java estándar, con input stream y output stream. Estos clases sólo soportan archivos situados en la carpeta de la aplicación, por lo que es necesario saber dónde están almacenadas las aplicaciones. Por defecto se almacenan en /data/data/nombre_paquete/files/nombre_fichero, pudiendo almacenarlas en la SDCard en las últimas versiones. Es importante ser conscientes de que algunos dispositivos tienen una memoria interna limitada, por lo que no deberíamos de abusar de este espacio guardando archivos de gran tamaño.

Existen tres alternativas para almacenar archivos locales:

MEMORIA INTERNA

El siguiente código muestra la clase "InputStreamReader()" que se usa para leer los archivos locales cuando estos están almacenados en la memoria interna. Hay que tener en cuenta que solamente se pueden leer aquellos ficheros para los que se tenga permiso.

Código Android

```
try
{
   BufferedReader fIn =
      new BufferedReader(
         new InputStreamReader(
            openFileInput("texto_ejemplo2.txt")));

   String texto = fIn.readLine();    // Cogemos la línea de fIn
   fin.close();    // Salimos para guardar la línea
}
catch (Exception e)
{
   Log.e("Error de fichero", "Error al leer el fichero");
}
```

A continuación se muestra el código de la clase "OutputStreamWriter()". En este caso hay que seleccionar el modo de acceso, que puede ser:

- MODE_PRIVATE (por defecto) acceso privado desde nuestra aplicación.
- MODE_APPEND para poder añadir datos a un fichero existente.
- MODE_WORLD_READABLE para permitir lectura a otras aplicaciones.
- MODE_WORLD_WRITABLE para permitir escritura a otras aplicaciones.

Código Android

```
try
{
   OutputStreamWriter fOut=
       new OutputStreamWriter(
          openFileOutput("texto_ejemplo.txt", Context.MODE_PRIVATE));

   fOut.write("Cualquier cosa");     // Escribimos "Cualquier cosa" en fOut
   fOut.close();                     // Cerramos la escritura en el fichero fOut
}
catch (Exception e)
{
   Log.e("Error de fichero", "Error al escribir el fichero.");
}
```

RECURSOS

Otra forma de guardar los archivos locales en la memoria interna es almacenándolos como un recurso en la carpeta "/res/raw" del proyecto, en eclipse dicha carpeta no se crea por defecto, hay que crearla. Es importante destacar que este método de almacenamiento es útil para ficheros que no vamos a modificar, ya que posteriormente serán de solo lectura. A continuación se muestra el código que implementa la manera de acceder a dichos archivos.

Código Android

```
try
{
   InputStream fRaw =
```

```
        getResources().openRawResource(R.raw.ejemplo_raw);
        //ejemplo_raw es el nombre del fichero
      BufferedReader bRIn =
        new BufferedReader(new InputStreamReader(fRaw));
      String linea = bRIn.readLine();
      fRaw.close();
}
catch (Exception e)
{
    Log.e("Error de fichero", "Error al leer fichero desde recurso");
}
```

MEMORIA EXTERNA

La última alternativa para almacenar archivos locales será la de guardarlos en la memoria externa, que generalmente será una tarjeta de memoria SD. Es importante destacar que en el caso de la memoria externa, y al contrario que con la memoria interna, puede no estar disponible ya sea porque no está presente o porque el dispositivo no la reconoce. Para ello tendremos que cercionarnos de que está disponible antes de utilizarla como sistema de almacenamiento, con ese fin se usa el método getExternalStorageStatus() que devuelve el estado de la memoria externa. Según se quiera leer o escribir en ella los estados que interesan son MEDIA_MOUNTED que informa que se puede escribir y leer en ella o MEDIA_MOUNTED_READ_ONLY que informa que está disponible con permisos de solo lectura.

El siguiente fragmento de código muestra una forma sencilla de verificar el estado de la memoria externa y almacenarlo en dos booleanos.

Código Android

```
boolean tarjetaDisponible = false;
boolean tarjetaEscritura = false;

//Comprobamos el estado de la memoria externa (tarjeta SD)
String estadoTarjeta = Environment.getExternalStorageState();

if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED))
```

```
{
    // Tarjeta disponible y habilitada para escritura
    tarjetaDisponible = true;
    tarjetaEscritura = true;
}
else if (estadoTarjeta.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
{
    // Tarjeta disponible y deshabilitada para escritura
    tarjetaDisponible = true;
    tarjetaEscritura = false;
}
else
{
    // Tarjeta NO disponible
    tarjetaDisponible = false;
    tarjetaEscritura = false;
}
```

En este caso hay que indicar en el Manifest que la aplicación necesita permisos para almacenar datos en memoria externa. Esto se hace añadiendo en el fichero Manifiest el siguiente código:

Código xml

```
<uses-permission android.permission.WRITE_EXTERNAL_STORAGE>
</uses-permission>
```

A continuación se muestra el código para escribir en los archivos locales que están almacenados en una tarjeta de memoria. Lo primero que hay que saber es la ruta absoluta de la tarjeta de memoria, por eso se usa el método getExternalStorageDirectory().

Código Android

```
try
{
    File ruta_tarjeta = Environment.getExternalStorageDirectory();
    File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
     OutputStreamWriter fOut =
        new OutputStreamWriter(
            new FileOutputStream(f));
     fOut.write("Smile! :D.");
    fOut.close();
}
catch (Exception e)
{
    Log.e("Error de ficheros", "Error al escribir fichero en la tarjeta.");
}
```

El siguiente código lee de los archivos locales almacenados en una tarjeta de memoria, es muy similar al de escritura.

Código Android

```
try
{
    File ruta_tarjeta = Environment.getExternalStorageDirectory();


    File f = new File(ruta_tarjeta.getAbsolutePath(), "ejemplo_tarjeta.txt");
    BufferedReader fIn =
        new BufferedReader(
            new InputStreamReader(
                new FileInputStream(f)));
    String linea = fIn.readLine();
    fIn.close();
```

```
}
catch (Exception e)
{
    Log.e("Error de ficheros", "Error al leer fichero de la tarjeta.");
}
```

BASES DE DATOS: SQLITE

Android tiene integrado en el propio sistema una API completa que nos permite manejar BBDD en SQLite. SQLite es un motor de bases de datos que se ha ido popularizando en los últimos años dado que maneja archivos de poco tamaño, no necesita ejecutarse en un servidor, cumple el estándar SQL-92 y, además, es de código libre.

El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), se guarda como un único fichero en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción. Esta es, al mismo tiempo, su gran virtud y su mayor inconveniente, ya que gracias a ello dispone de unas latencias muy bajas, pero también impide el acceso múltiple a la base de datos.

Centrándonos en el lenguaje que abordamos en este tutorial, la manera más fácil de acceder a la información de una base de datos local es creando una clase que herede de SQLiteOpenHelper sobre la cual tendremos que adaptar/sobreescribir los métodos proporcionados para obtener la funcionalidad con la base de datos deseada. Básicamente en esta clase definimos los atributos de nuestra base de datos y el comportamiento ante creación y actualización de la misma. Los métodos que deberemos sobrescribir serán onCreate() y onUpgrade(), además de la constructora, donde podremos incluir todo lo que creamos necesario.

A modo de ejemplo, vamos a crear una base de datos de una entidad Domicilio que incluirá como atributos: calle, ciudad, CP y número. A continuación se muestra el código de la clase DomicilioSQLiteHelper:

Código Android

```
public class DomicilioSQLiteHelper extends SQLiteOpenHelper {


    //Aqui creamos el String que creará la base de datos en el onCreate
    String creaBD= "CREATE TABLE Domicilio (calle TEXT, ciudad TEXT, CP
  INTEGER, numero INTEGER)";
```

```
    public DomicilioSQLiteHelper(Context context, String nombre,
                                 CursorFactory factory, int version) {
        super(context, nombre, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //Aqui se crea la BD, se llamaría solo cuando no exista
        db.execSQL(sqlCreate);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int a, int b) {
        /*Utilizamos como opción de actualización la de borrado de la tabla anterior, para luego crearla vacia con la nueva versión*/
        db.execSQL("DROP TABLE IF EXISTS Domicilio"); //Borrado anterior
        db.execSQL(sqlCreate); //Creación nueva
    }
}
```

Es interesante destacar en qué momento se usarán los métodos onCreate() y onUpgrade() que aparecen en el código anterior:

- onCreate() se ejecutará de maneara automática cuando se necesite crear la base de datos, lo que se reduce a cualquier llamada o referencia a la base de datos mientras esta todavía no exista.
- onUpgrade() será llamado automáticamente cuando se produzca un cambio en la estructura de la base de datos, como podría ser la inclusión/eliminación de un campo en una tabla.

Por otro lado hay que tener en cuenta que en el código de ejemplo anterior se ha borrado la tabla anterior y creado una tabla nueva vacía, sin embargo, en la mayoría de los caso es necesario migrar los datos del modelo anterior al nuevo.

A continuación se muestra el código de un ejemplo sencillo que abre una base de datos para escritura y crea un par de registros. Como la clase hereda de SQLiteOpenHelper, se pueden

usar los métodos getReadableDatabase() o getWritableDatabase() para acceder a la base de datos en modo lectura o lectura/escritura.

Código Android

```
Int version = 1;
String nombreDB = "DomiciliosDB";
/****** Nuestro código anterior aquí *******/
      DomicilioSQLiteHelper domicilioHelper =
         new UsuariosSQLiteHelper(this, nombreDB, null, version);

      // Abrimos la Base de datos, en casa de que no existiera, se crearía ahora
      SQLiteDatabase db = domicilioHelper.getWritableDatabase();

         //Insertamos 2 domicilios para inicializar la tabla
         db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
                  VALUES ('C/Mayor', 'Madrid', 28001, 13)");
         db.execSQL("INSERT INTO Domicilio(calle, ciudad, CP, numero)
                  VALUES ('Avda. Grande', 'Ibiza', 26050, 45)");

         //Cerramos la base de datos ya actualizada
         db.close();

/****** Resto de código *******/
```

Pese a haber aparecido antes, no se ha comentado el uso del método execSQL(String query). Como el lector habrá podido imaginar, este método recibe Strings de los query que se pueden ejecutar sobre la base de datos y los ejecuta. La intención de este tutorial, no es ni mucho menos enseñar la utilización de bases de datos con el estandar de SQLite, para obtener información sobre él por favor, visite la página oficial de SQLite y encontrará información sobre su sintaxis:

http://www.sqlite.org/lang.html

A pesar de lo citado anteriormente, para no quedarnos sólo con estas breves pinceladas sobre la sintaxis de SQLite, a continuación se muestran las funcionalidades más comunes de una base de datos en modo escritura:

- Insertar un registro

  db.execSQL("INSERT INTO Usuarios (usuario,email) VALUES ('usu1','usu1@email.com') ");

- Eliminar un registro

  db.execSQL("DELETE FROM Usuarios WHERE usuario='usu1' ");

- Actualizar un registro

  db.execSQL("UPDATE Usuarios SET email='nuevo@email.com' WHERE usuario='usu1' ");

El siguiente código muestra las funcionalidades de los cursores. Los cursores simplemente sirven para recorrer el resultado de una base de datos como si de un iterador se tratase. Como se van a ejecutar algunas querys sobre la clase domicilio, la base de datos de los domicilios se tiene que abrir sólo con permisos de lectura.

Código Android

```
Int version = 1;
String nombreDB = "DomiciliosDB";
/****** Nuestro código anterior aquí *******/
DomicilioSQLiteHelper domicilioHelper = new UsuariosSQLiteHelper(this, nombreDB, null, version);
 // Abrimos la Base de datos, en casa de que no existiera, se crearía ahora
SQLiteDatabase db = domicilioHelper.getReadableDatabase();
 Cursor c = db.rawQuery("SELECT ciudad, CP FROM Domicilio WHERE calle= "C/Mayor");
//Obtenemos los indices de las columnas
int ciudadIndex = mCursor.getColumnIndexOrThrow("ciudad");
int CPIndex = mCursor.getColumnIndexOrThrow("CP");
// Posicionamos el cursor al principio de la lista
if (c.moveToFirst()) {
    do {
    //Obtenemos en nuestras variables los datos del registro que está leyendo.
        String ciudad= c.getString(ciudadIndex);
        int CP = c.getString(CPIndex);
    } while(c.moveToNext());
    /*Lo seguimos adelantando mientras tengamos registros que leer,
```

```
        Aunque parezca extraño, por la falta de uso, es muy común utilizar aqui la
        estrucutra DO- WHILE*/
    /****** Resto de código *******/
```

CONTENT PROVIDERS

Los Content Providers son el mecanismo que tiene Android para comunicar datos entre distintas aplicaciones. Funcionalmente suelen ser muy parecidos a las bases de datos en SQLite que se han descrito en el punto anterior. El propio Android trae, desde sus versiones 2.0+, incluidos la agenda , los SMS y el listado de llamadas mediante el método de los Content Providers, simplemente una aplicación debe acceder a la URI donde se encuentra el Content Provider y una vez tiene acceso pedirle los datos que necesita.

Al igual que para las bases de SQLite para los Content Providers hay que crear una clase que herede de ContentProvider y declararla en el manifest como se muestra a continuación.

AndroidManifest.xml

```
<application>....
<provider android:name="DomiciliosProvider"
   android:authorities="com.pruebasandroid.ejemplo" />
<application/>
```

La facilidad de su uso reside en tener una base de datos que realice las mismas funciones que los métodos que hay que sobreescribir. Para llamarlos basta con hacer un buen uso de las URIs. A continuación se muestra el código, a modo de ejemplo, de DomiciliosProvider:

Código Android

```
package com.pruebasandroid.ejemplo;
…
public class DomiciliosProvider extends ContentProvider {
      //Establecemos como una constante la URI para aeder a estos datos
   private static final String uri = "content://com.pruebasandroid.ejemplo/domicilios";
       //Uri del String anterior
   public static final Uri CONTENT_URI = Uri.parse(uri);
```

```
    //Necesario para UriMatcher
  private static final int DOMICILIOS = 1;
  private static final int DOMICILIOS_ID = 2;
  private static final UriMatcher uriMatcher;
    //Clase interna para declarar las constantes de columna
  public static final class Clientes implements BaseColumns
  {
    private Clientes() {}
        // Inicializamos las columas, para poder acceder a ellas con un mnemotécnico
    public static final String COL_CALLE = "calle";
    public static final String COL_CP = "cp";
    public static final String COL_NUMERO = "numero";
    public static final String COL_CIUDAD = "ciudad";
  }
    //Los datos referentes a la base de datos, que creamos en el ejemplo anterior
  private UsuariosSQLiteHelper helper;
  private static final String NOMBRE_BD = "DomicilioDB";
  private static final int VERSION = 1;
  private static final String TABLA_DOMICILIOS = "Domicilios";
    //Inicializamos el UriMatcher
  static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("com.pruebasandroid.ejemplo", "domicilios",    DOMICILIOS);
    uriMatcher.addURI("com.pruebasandroid.ejemplo",                  "domicilios/#",
DOMICILIOS_ID);
  }
  /*Sobreescribimos los métodos de ContentProvider*/
  @Override
  public boolean onCreate() {
    helper = new UsuariosSQLiteHelper(
                                this.getContext(), NOMBRE_DB, null, VERSION);
```

```
    return true;
}
/*Aqui devolvemos el cursor de nuestra BD de domicilios*/
@Override
public Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs, String sortOrder) {

    /*Construimos un where si se refiere a un id concreto */
    String where = selection;
    if(uriMatcher.match(uri) == DOMICILIOS_ID){
        where = "_id=" + uri.getLastPathSegment();
   }
    /*Escritura y lectura y devolvemos el cursor de nuestra base de datos*/
    SQLiteDatabase db = helper.getWritableDatabase();

    return (Cursor)db.query(TABLA_DOMICILIOS, projection, where,
                            selectionArgs, null, null, sortOrder);
}
@Override
public Uri insert(Uri uri, ContentValues values) {
    /*para escribir, insertamos en la base de datos que nos proporciona*/
    long id = 1;
    SQLiteDatabase db = helper.getWritableDatabase()
    id = db.insert(TABLA_DOMICILIOS, null, values);

    /*Devolvemos su correspondiente Uri*/
    Uri newUri = ContentUris.withAppendedId(CONTENT_URI, id);
    return newUri;
}
@Override
public int update(Uri uri, ContentValues values,
```

```
                    String selection, String[] selectionArgs) {
            /* Aqui el código para el update*/
    }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        /* Aqui el código para el delete*/
    }

    @Override
    public String getType(Uri uri) {
      int tipo = uriMatcher.match(uri);
      switch (tipo)
      {
          case DOMICLIOS:
                  return "vnd.android.cursor.dir/vnd.pruebasandroid.domicilio";
          case DOMICILIOS_ID:
                  return "vnd.android.cursor.item/vnd.pruebasandroid.domicilio"";
          default:
                  return null;
      }
    }
}
```

DATOS EN RED

A pesar de lo que hemos comentado sobre SQLite anteriormente, muchas son las aplicaciones que necesitan la integración con una base de datos existente en red como un MySQL. La diferencia principal respecto a los datos que se pueden manejar con SQLite reside, en que estos datos, se guardaban en un archivo de forma local, mientras que con las BBDD que podemos manejar a través de la red, tenemos cambios de otros usuarios de manera inmediata.

La mejor forma de conectar nuestro terminal con una base de datos de manera remota es mediante el uso de un servicio web. En este tutorial se va a explicar cómo crear, de manera sencilla, un servicio web en php que proporcione datos desde una base de datos MySQL, y cómo interactuar

desde Android con ese servicio. Este tipo de servicio es simplemente un ejemplo, si el lector conoce o prefiere utilizar cualquier otro tipo de lenguaje en la creación del servicio web es una opción totalmente válida. Se ha elegido el lenguaje php porque es un lenguaje muy sencillo para el manejo de bases de datos.

A modo de ejemplo se va crear una base de datos que tiene una tabla con datos de personas. A cada persona se le asigna los siguientes atributos: DNI, nombre, sexo y fecha de nacimiento. Lo primero que hay que hacer es entrar en la consola de SQL de nuestro servidor y escribir el siguiente código:

Código SQL.

```
CREATE TABLE `nuestraBD`.`persona` (
`DNI` VARCHAR( 10 ) NOT NULL ,
`nombre` VARCHAR( 30 ) NOT NULL ,
`sexo` VARCHAR( 10 ) NOT NULL ,
`fechaNacimiento` DATE NOT NULL
)
```

Con esto se ha creado en la base de datos, la entidad persona. Sólo queda introducir los cuatro atributos que tiene cada persona, esto se hace escribiendo el siguiente código:

Código SQL.

```
INSERT INTO `nuestraBD`.`persona` (
`DNI` ,
`nombre` ,
`sexo` ,
`fechaNacimiento`
)
VALUES (
'000000000X', 'Jorge', 'Hombre', '1980-11-20'
), (
'11111111Y', 'Francisco', 'Hombre', '1977-09-10'
), (
'222222222Z', '', 'Mujer', '1987-01-15'
);
```

Una vez creada la base de datos, hay que crear un servicio web que obtenga todos los datos de la tabla. El siguiente código muestra cómo hacerlo. Obviamente el servicio web aceptará cualquier sintaxis y restricción que le pidamos como al propio SQL (Where, Grouopby, limit ,sort....)

Código PHP.

```
<?php
mysql_connect("host","username","password");
mysql_select_db("nuestraBD");

$pregunta = "select * from persona"

$sql=mysql_query($pregunta);
while($row=mysql_fetch_assoc($sql))
        $output[]=$row;
print(json_encode($output));
mysql_close();
?>
```

El siguiente paso es publicar en la web el código anterior y renombrarlo con "personas.php". Hay que recordar que los datos de la conexión especificados por "host","username","password", deben corresponder con los de la base de datos que el usuario requiere para su aplicación. En el código anterior, simplemente se ha lanzado una query que devuelve todos los datos de la tabla persona y después se ha ido creando un array que contiene cada entrada de la propia tabla. Como se puede observar en el código, al final siempre hay que cerrar la conexión, pero antes de ello hay que escribir print(json_encode($output)); para poder obtener los datos como salida en el estandar JSON.

JSON (JavaScript Object Notation) es un lenguaje de etiquetas como XML que se utiliza como sustituto para intercambio de contenidos. Dada su sencillez permite crear parsers mucho más sencillos que en XML. En el ejemplo anterior, la estructura que devuelve JSON es la siguiente:

Respuesta JSON

```
[{"DNI":"000000000X","nombre":"Jorge","sexo":"Hombre","fechaNacimiento":"1980-11-20"},

{"DNI":"11111111Y","nombre":"Francisco","sexo":"Hombre","fechaNacimiento":"1977-09-10"},

{"DNI":"222222222Z","nombre":"Patricia","sexo":"Mujer","fechaNacimiento":"1987-01-15"}]
```

Una vez se tiene la base de datos y el servicio web, lo primero que hay que hacer es introducir en el *Manifest* permisos para que la aplicación pueda acceder a Internet, de esta forma podremos concertar con nuestro servicio Web:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Lo siguiente es crear una función que consiga conectar con el servicio web y pueda enviar/recibir datos. De esta manera, se le podrá pedir al servicio Web datos para rellenar los campos DNI, nombre, sexo y fechaNac de la tabla de la base de datos. Con ello ya podremos usarlas en nuestra aplicación, mostrarlas en un ListView o todo aquello que veamos necesario. El siguiente código muestra como implementar dicha función.

Código Android

```
public class Persona extends Activity {

@Override
public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.main);
try{
    HttpClient httpclient = new DefaultHttpClient();

    HttpPost httppost = new HttpPost("URL/persona.php");

    HttpResponse response = httpclient.execute(httppost);

    HttpEntity entity = response.getEntity();

    InputStream is = entity.getContent();

    }catch(Exception e){

       System.out.println("Error en la conexión");

       // En esta caputra de excepción podemos ver que hay un problema con la

       // conexión e intentarlo más adelante.

    }
try{
     BufferedReader reader = new BufferedReader(new InputStreamReader

   (is,"iso-8859-1"),8);


         StringBuilder  sb = new StringBuilder();

         sb.append(reader.readLine() + "\n");
         String line="0";
         while ((line = reader.readLine()) != null) {
                  sb.append(line + "\n");
```

```
            }
            is.close();
            String cadena =sb.toString();
            }catch(Exception e){
                System.out.println("Error al obtener la cadena desde el buffer");
            }
//Creamos los atributos de nuestra clase persona
String DNI; String nombre; String sexo; String fechaNac;

try{
    JSONArray jsonArray = new JSONArray(cadena);
    JSONObject jsonObject=null;
    for(int i=0;i<jsonArray.length();i++){
        jsonObject= jsonArray .getJSONObject(i);
        DNI=jsonObject.getString("DNI");
        nombre=jsonObject.getString("nombre");
        sexo=jsonObject.getString("sexo");
        fechaNac=jsonObject.getString("fechaNacimiento");
      }
    }
    catch(JSONException e1){
          Toast.makeText(getBaseContext(), "No se encuentran los datos"
,Toast.LENGTH_LONG).show();
    } catch (ParseException e1) {
                        e1.printStackTrace();
        }
}
}
  }
```

Ahora bien, hasta lo que hemos visto, solo podemos pedirle al servicio Web datos pero ¿qué pasa si queremos introducir algún tipo de información? La respuesta es muy sencilla en php una de las cosas más comunes es pedir el usuario y la contraseña para acceder a la base de datos, o simplemente los datos a introducir en un *insert* o cualquier otro dato que necesitemos desde php. Para obtener datos externos desde PHP utilizamos $_REQUEST:

Código PHP

```
<?php
        $host =  $_REQUEST[host];
        $user =  $_REQUEST[user];
```

```
        $password = $_REQUEST[password];
        mysql_connect($host , $user, $password);
//    ….....Nuestro código intermedio aqui.....
        mysql_close();
?>
```

Introducir estos datos desde Android también es sencillo. En el código del ejemplo anterior lo que hay que hacer es añadir al HttpPost las siguientes linéas:

Código Android

```
//Creamos un ArrayList de Nombre Valor
// e introducimos los datos que deseamos pasar al web service
ArrayList<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("host","url de nuestro host"));
nameValuePairs.add(new BasicNameValuePair("user","usuario de la BD"));
nameValuePairs.add(new BasicNameValuePair("password","contaseña"));
// Añadimos los valores al HttpPost
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost =  HttpPost("URL/persona.php");
httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
```

6. MAPAS Y GPS

Álvaro Borrego – Francisco Hernández – David Palomero

USO DE MAPAS

Para poder usar los mapas, Android provee de una librería externa que se encuentra en el paquete ***com.google.maps***

¿COMO OBTENER LA API KEY PARA USAR GOOGLE MAPS?

Para poder acceder a los datos desde el MapView es necesario registrarse en el servicio de Google Maps y aceptar los términos de uso. De esta manera se obtendrá una clave alfanumérica que nos dará acceso.

El registro para obtener la clave consta de dos partes:

6. Registrar la huella digital MD5 de la aplicación para que pueda acceder a los datos de Google Maps.
7. Agregar una referencia a la clave en cada MapView (en el XML o en código).

INFORMACIÓN GENERAL

Para asegurar que los aplicaciones utilizan los datos de manera adecuada, el MapView necesista una clave para poder usar la API. Esta clave es una secuencia alfanumérica que identifica la aplicación y el desarrollador. Sin esta clave, el MapView no podrá descargar los datos de los mapas.

Cada ApiKey de Google Maps está asociada a un certificado en concreto. Y cada MapView debe hacer referencia a un clave de la API (ApiKey).

Varias vistas Map puede referirse al mismo o distintos si se han registrado varios certificados a la misma aplicación.

¿CÓMO OBTENER LA HUELLA DIGITAL MD5 DEL CERTIFICADO?

Para generarla necesitamos usar la herramienta ***keytool*** del SDK de java. Los parámetros para el ***keytool*** se muestran en la siguiente tabla:

Tabla 1. Parámetros de Keytool

Opciones Keytool	Descripción
`-list`	Muestra la huella MD5 del certificado
`-keystore <keystore-name>.keystore`	El nombre del keystore que contiene la clave
`-storepass <password>`	Una clave para el keystore.
`-alias <alias_name>`	The alias for the key for which to generate the MD5 certificate fingerprint.
`-keypass <password>`	The password for the key. As a security precaution, do not include this option in your command line unless you are working at a secure computer. If not supplied, Keytool prompts you to enter the password. In this way, your password is not stored in your shell history.

La Figura 1 muestra un ejemplo de la obtención de la clave.

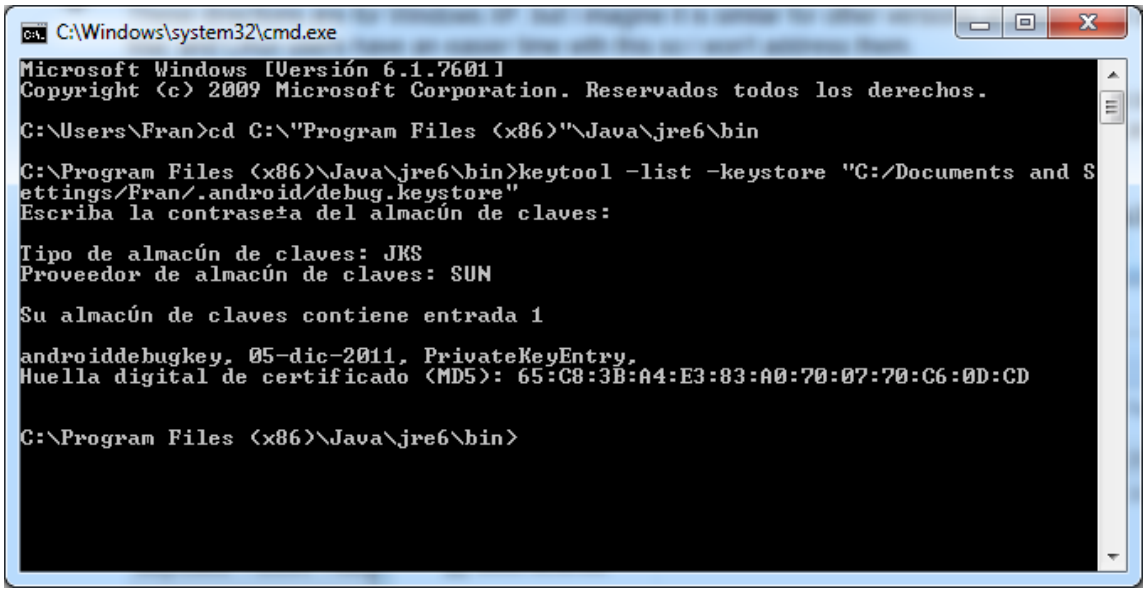


Figura 1. Ejemplo de la obtención de la clave

REGISTRAR LA HUELLA MD5 EN EL SERVICIO DE GOOGLE MAPS

Una vez obtenida la huella digital hay que acceder a la siguiente dirección Web:

http://code.google. com/android/maps-api-signup.html

La página muestra la información de la Figura 2. Hay que leer las condiciones, pegar la huella digital, y dar al botón para generar la clave. Entonces aparecerá una ventana como la que se ve en la Figura 3, donde se encuentra la clave.

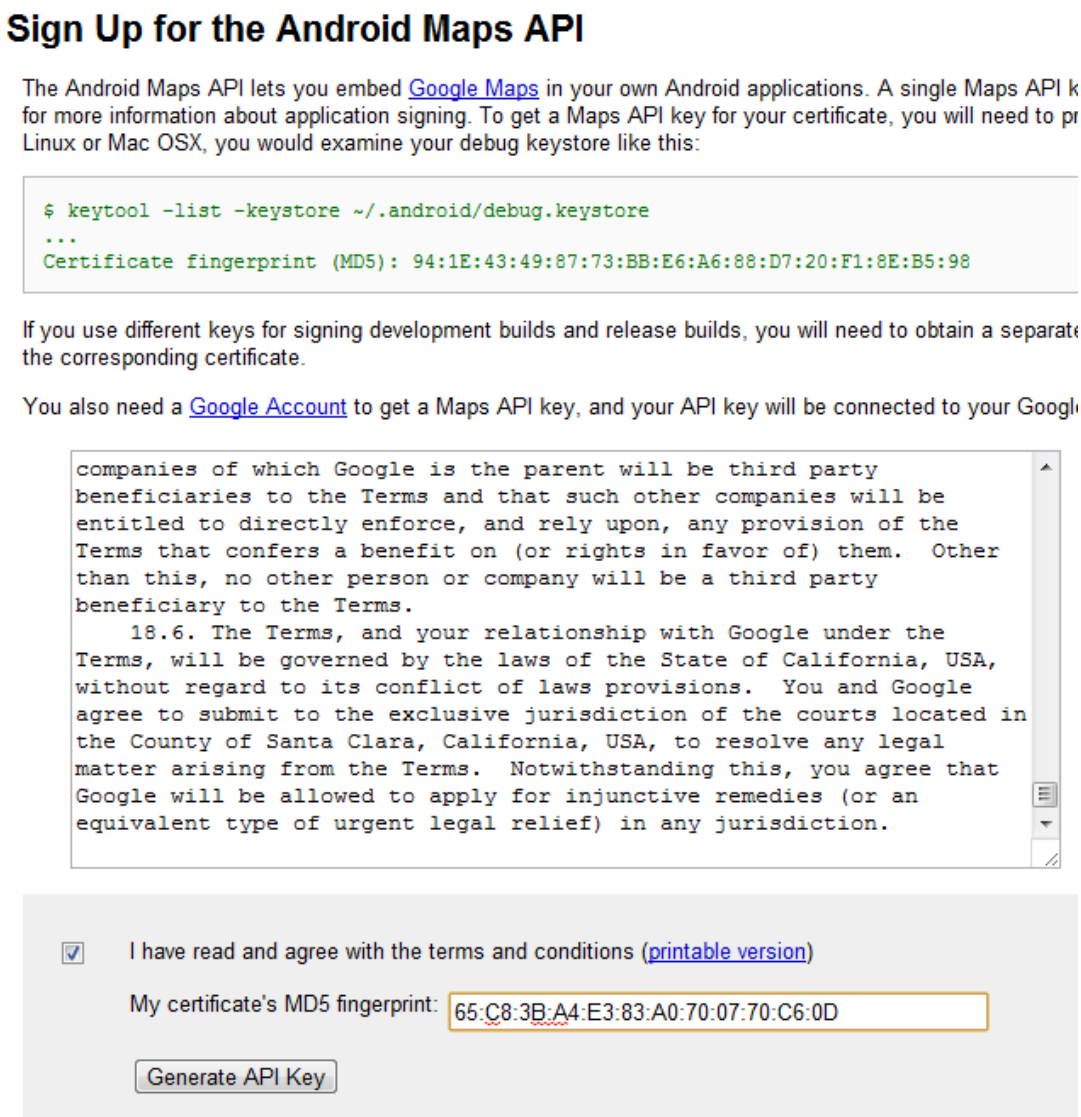


Figura 2. Registro huella MD5 en Google Maps

Figura 3. Api de Google Maps

AÑADIR LA CLAVE DEL API DE ANDROID MAPS A NUESTRA APLICACIÓN Y AÑADIRLA AL LAYOUT

Una vez generada la clave, hay que añadirla a la aplicación. Esto se puede hacer de dos formas:

- Desde el XML del layout, mediante el siguiente código:

```
<com.google.android.maps.MapView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="example_Maps_ApiKey_String"
 />
```

- Desde una actividad, añadiendo la siguiente línea:

```
MapView mMapView = new MapView(this, "example_Maps_ApiKey_String");
```

HABILITAR EL USO DEL MAPVIEW

Para habilitar el uso del MapView hay que añadir la referencia a la librería externa com.google.android.maps en el Manifest como muestra el siguiente código. Será un hijo del elemento **<applicationd>:**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="es.ucm.fdi.localyza"
      android:versionCode="1"
      android:versionName="1.0">
...
    <application android:icon="@drawable/icon"
                 android:label="@string/app_name"
```

```
        android:theme="@style/OrangeTheme">
    <uses-library android:name="com.google.android.maps" />
```

Cuando la aplicación esté lista para ser distribuida en el Market, debermos modificar la clave de los mapas por la generada con el keystore con el que hemos firmado nuestra aplicación.

MOSTRAR EL ZOOM VIEW

Para añadir los controles de Zoom, hay que llamar a la función displayZoomControls usando el siguiente código:

```
private MapView mapa;
mapa = (MapView)findViewById(R.id.mapa);
mapa.displayZoomControls(true);
```

La Función onKeyDown() descrita a continuación, es la que permite el manejo del zoom.

```
public boolean onKeyDown(int keyCode, KeyEvent event)
{
      MapController mc = mapa.getController();
      switch (keyCode){
            case KeyEvent.KEYCODE_3:
                  mc.zoomIn();
                  break;

            case KeyEvent.KEYCODE_1:
                  mc.zoomOut();
                  break;

       }
      return super.onKeyDown(keyCode, event);

}
```

CAMBIAR LAS VISTAS DEL MAPA

Activación de la vista satélite (ver Figura4): `mapa.setSatellite(true);`

Activación de la vista de calles (ver Figura4): `mapa.setStreetView(true);`

Figura 4. Vista satélite Google Maps

Figura 5. Vista de calles Google Maps

SITUAR LA POSICIÓN DEL MAPA EN UN PUNTO CONCRETO

Para situar la posición del mapa en un punto concreto hay que definir la longitud y latitud, esto se consigue creando un GeoPoint y, mediante el MapController, posicionando la vista en las coordenadas. El siguiente código muestra como hacerlo:

```
mapa.setBuiltInZoomControls(true);
Double latitud = 40.452723*1E6;
Double longitud = -3.733253*1E6;
```

```
GeoPoint loc =
            new GeoPoint(latitud.intValue(), longitud.intValue());
MapController controlMapa = mapa.getController();
controlMapa.animateTo(loc);
```

AÑADIR CAPAS A LOS MAPAS

Primero se define la capa Overlay que pintará el icono.

```
class MapOverlay extends com.google.android.maps.Overlay
{
      …
}
```

Y hay que redefinir el método draw como indica el siguiente código:

```
 public boolean draw(Canvas canvas, MapView mapView, boolean shadow,
long when)
 {
      super.draw(canvas, mapView, shadow);
      //---transformamos el geopoint a pixeles---
      Point screenPts = new Point();
      mapView.getProjection().toPixels(loc, screenPts);
      //--- añadimos el marcador---
      Bitmap bmp = BitmapFactory.decodeResource(
            getResources(), R.drawable.point_mor);
     canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50, null);
      return true;
 }
```

Por último hay que crear una lista de MapsOverlays e indicar sobre que mapView ha de pintarse. El siguiente código muestra como hacerlo.

```
//---Añadimos el marcador de la localización---
      MapOverlay mapOverlay = new MapOverlay();
      List<Overlay> listOfOverlays = mapa.getOverlays();
      listOfOverlays.clear();
      listOfOverlays.add(mapOverlay);
      mapa.invalidate();
```

GPS

Actualmente, existe una gran cantidad de aplicaciones basadas en localización, y día a día siguen aumentando. Existen treinta y un satélites sin nada mejor que hacer que proveernos de estos servicios.

SOLICITUDES DE PERMISOS

Para obtener permiso para poder usar geolocalización hay que añadir en el Manifest las siguinetes líneas.

```
<manifest ... >
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

OBTENIENDO LAS ACTUALIZACIONES DE LA LOCALIZACIÓN

Para obtener la localización, Android funciona por medio de la devolución de la llamada. Se recibe por medio de las actualizaciones del LocationManager, llamando al método requestLocationUpdates, y pasándole un LocationListener.

```
// Obtención de una referencia al LocationManager

        LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

//Definición del listener que responde a las actuzalizaciones de la
localización.

        LocationListener locationListener = new LocationListener() {

            public void onStatusChanged(String provider, int status,
Bundle extras) {}

            public void onProviderEnabled(String provider) {}

            public void onProviderDisabled(String provider) {}

                    public void onLocationChanged(Location location) {

                          //Que hacer con la nueva localización
                    }
          };

// Registrar el listener con el LocationManager para recibir
actualizaciones

locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDE
R, 0, 0, locationListener);
```

En esta última línea, tenemos que indicar si queremos usar la red o los GPS como proveedor sustituyendo "LocationManager.NETWORK_PROVIDER" por "LocationManager.GPS_ PROVIDER".

Los otros dos parámetros son:

- minTime: es el minimo intervalo de tiempo para las notificaciones en milisegundos. Se usa para evitar hacer demasiadas peticiones al servicio de localización y ahorrar batería.
- minDistance: el minimo intervalo de distancia para las notificaciones, en metros.

Para detener las actualizaciones, tenemos que eliminar el listener que previamente habíamos indicado al LocationManager:

```
locationManager.removeUpdates(locationListener);
```

OBTENER LA ÚLTIMA LOCALIZACIÓN CONOCIDA

En ocasiones, podemos querer una rápida localización y en un primer instante, puede ser útil utilizar las últimas coordenadas válidas. Podemos obtenerlas de la siguiente forma:

```
locationProvider = LocationManager.NETWORK_PROVIDER;
        Location lastKnownLocation =
locationManager.getLastKnownLocation(locationProvider);
```

GEOCODER

Es una clase que permite transformar una dirección o descripción de un lugar en las coordenadas (longitud y latitud). La geo codificación inversa es el proceso de transformar las coordenadas en una dirección parcial. La cantidad de detalles obtenidos puede variar según el lugar.

¿CÓMO OBTENER EL LUGAR A PARTIR DE UNAS COORDENADAS?

```
Geocoder gc=new Geocoder(getApplicationContext());
List<Address> address= gc.getFromLocation (latitude, longitude,
                        maxResults);
```

¿CÓMO OBTENER LUGARES A PARTIR DE UNA DESCRIPCIÓN?

```
Geocoder gc=new Geocoder(getApplicationContext());
List<Address> address=gc.getFromLocationName(locationName,
maxResults);
```

7. TELEFONÍA

Manuel Báez – Jorge Cordero – Miguel González

MENSAJES DE TEXTO

La posibilidad de enviar y recibir mensajes de texto provocó una revolución en los teléfonos móviles, siendo una de las principales formas de comunicación durante la primera década del SXXI hasta la llegada del internet móvil. Como es de esperar Android nos da la posibilidad de enviar SMS. La encargada de esto es la clase Android.telephony.SmsManager que soporta tanto GSM(*Groupe spéciale mobile*) como CDMA(*Code Division Multiple Access*). Dicha clase tiene únicamente cinco métodos públicos que son los siguientes.

- ArrayList<String> divideMessage(String text): Su función es la de recibir como parámetro un String text y convertirlo en un ArrayList de Strings. El tamaño de estos Strings será menor o igual al máximo permitido por un SMS
- static SmsManager getDefault(): Devuelve la instancia por defecto de SmsManager
- void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[ ] data, PendingIntent sentIntent, PendingIntent deliveryIntent): Es el método que se encarga de enviar el SMS. Le pasamos como parámetro la dirección de destino, el puerto de destino, el SMS en sí y dos PendingIntent que sirven para el envío y el acuse de recibo.
- void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents): En el caso de tener un SMS de longitud mayor que lo permitido para uno tendremos que usar este método. Su funcionamiento es el mismo que el de sendDataMessage, la diferencia es que hay un array list para cada PendingIntent así como para cada parte(parts).
- void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Para enviar un SMS sirviéndonos de dichos métodos hay que tener en String text el texto que se quiere enviar, y en String phoneNumber el número al que se va a enviar. También se necesita tener un PendingIntent , esto se obtiene del siguiente modo:

```
PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, SMS.class), 0);
```

A continuación hay que usar el método SmsManager.getDefault() para tener la instancia por defecto de SmsManager, que llamaremos sms. Existen dos opciones, que el texto quepa en un SMS o que no. En el caso de que quepa, la solución es más sencilla, solamente hay que llamar al método sms.sendTextMessage(phoneNumber, null, text, pi, null). Se han dejado los campos scAddress y deliveryIntent como null, para que tome un valor por defecto.

En el caso de que text no quepa en un único SMS hay que dividir el texto del SMS en varios mensajes, para ello se utiliza el método sms.divideMessage(text) y se guarda el resultado obtenido en arrayText. En este caso para a enviar el mensaje al número deseado se usa el método sms.sendMultipartTextMessage(phoneNumber, null, arrayText, pi, null) y con esto se habrá enviado el mensaje múltiple.

Es importante destacar que en el caso de los sms y mms, al ser información que ha de ser accesible desde cualquier aplicación, es necesario que se almacenen como Content Providers, y desde la versión 2.0+ es así. Están almacenados en la ruta /data/data/com.android.providers/telephony/databases/mmssms.db, y podemos ver cómo acceder a ellos en el punto del manual de los Content Providers.

Es importante destacar que el URI de los sms es el siguiente:

Código Android

```
String url = "content://sms/";
Uri uri = Uri.parse(url);
getContentResolver().registerContentObserver(uri, true, new MyContentObserver(handler));
```

LLAMADAS A TELÉFONO

La razón por la que se crearon los teléfonos móviles era la de poder tener una línea no fija desde la que pudiéramos hacer llamadas desde cualquier punto, siempre y cuando haya cobertura por supuesto, sin necesidad de cables ni nada por el estilo. Por ello es lógico suponer e incluso obvio que Android nos da la posibilidad de llamar desde una aplicación propia. A continuación se va a explicar realizar dichas llamadas. Para poder hacerlo primero hay que explicar qué es un Intent y cómo usarlos con un startActivity.

Un Intent (intento) es un paquete, dicho paquete contiene información del componente que reciba el intento. La constructora de Intent que se va a utilizar en este caso es de la forma Intent(String action, Uri uri) siendo el primer argumento, action, la acción a realizar, y el segundo, uri, el dato sobre el que realizar dicha acción. Por su parte startActivity(Intent), hay otras formas de llamar a startActivity pero en esta caso nos interesa ésta, lo que hace es intentar ejecutar la actividad determinada por Intent, poniéndola en la cima de la pila de actividad.

A continuación se puede ver un fragmento de código que implementa el AlertDialog que aparecerá cuando se pulsa sobre un contacto. Dicho AlertDialog tiene varias opciones de contacto entre ellas está la de "Llamar" cuyo código está enmarcado en rojo.

Código Android

```
private void launchContactOptions(final ContactListItem  contacto) {
    // Elementos del Dialog
    final String items[] = {"Llamar","Otro"};
    AlertDialog.Builder dialog = new AlertDialog.Builder(ContactosActivity.this);
    dialog.setTitle("Opciones de contacto");
    dialog.setItems(items, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface d, int choice) {
            switch (choice) {
            case 0:     // Éste es el código que nos interesa
                //Elegida opción "Llamar" del AlertDialog
                String url = "tel:" + contacto.getNum();
                Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(url));
                ContactosActivity.this.startActivity(intent);
                break;
            case 1:
            //Elegida opción "Otro" del AlertDialog


            break;
            default:
            break;
            }
        }
    });

    dialog.show();
}
```

Respecto al Registro de llamadas (CallLog) e información de las llamadas. Para obtener la lista de llamadas hay que llamar al android.provider.CallLog el cual se encarga de guardar un registro de las llamadas tanto recibidas como realizadas. Algunos campos importantes del registro CallLog se pueden ver en la siguiente tabla:

Campo	**Descripción**

android.provider.CallLog.Calls.NUMBER	El número involucrado
android.provider.CallLog.Calls.TYPE	Tipo de llamada. Entrante (1) o saliente (0).
android.provider.CallLog.Calls.CACHED_NAME	Nombre asociado con el número en caso de existir en la agenda.
android.provider.CallLog.Calls.DATE	Fecha de la llamada
android.provider.CallLog.Calls.DURATION	Duración de la llamada en segundos

El siguiente código implementa la consulta de llamadas salientes de duración superior a 0 segundos.

Código Android

```
ArrayList<String[]> result = new ArrayList<String[]>();


  // Consulta SQL
                String[] strFields = {
                      android.provider.CallLog.Calls.NUMBER,
                      android.provider.CallLog.Calls.TYPE,
                      android.provider.CallLog.Calls.CACHED_NAME,
                      android.provider.CallLog.Calls.DATE,
                      android.provider.CallLog.Calls.DURATION
                      };
//Condición llamadas salientes:
        String where = new String("1=" + android.provider.CallLog.Calls.TYPE + "");
        where += " AND " + android.provider.CallLog.Calls.DURATION + ">0";
        String strOrder = android.provider.CallLog.Calls.DATE + " DESC";
                Cursor mCursor = getContentResolver().query(
                      android.provider.CallLog.Calls.CONTENT_URI,
```

```
                    strFields,
                    where,
                    null,
                    strOrder
                    );
        // Iteramos sobre la lista de resultados a través del cursor.
                int nameIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.CACHED_NAME);
                int numberIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.NUMBER);
                int typeIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.TYPE);
                int dateIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DATE);
                int durIndex =
mCursor.getColumnIndexOrThrow(android.provider.CallLog.Calls.DURATION);
                if (mCursor.moveToFirst()) {
                   do {
                      String name = mCursor.getString(nameIndex);
                      String number = mCursor.getString(numberIndex);
                      String type = mCursor.getString(typeIndex);
                      String date = mCursor.getString(dateIndex);
                      String dur = mCursor.getString(durIndex);
                      result.add(new String{type, name, number, date, dur});

                   } while (mCursor.moveToNext());
                }
```

Para acceder a la agenda se usa la clase android.provider.**ContactsContract**, esta clase está disponible desde la versión 2.0+ (API Level 5). Para poder acceder a los contactos hay que añadir permisos a la aplicación en el archivo AndoridManifest.xml como se muestra a continuación.

Código Andorid Manifest.xml

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Para obtener datos de la agenda de contactos se almacena una instancia del ContentResolver y sobre esta se ejecuta el método query(). La llamada al método query() tiene como primer argumento la URI que identifica el conjunto de datos sobre el que se quiere ejecutar el método. Como segundo argumento se introduce un array con el número de columnas que se quieren obtener. Adicionalmente se pueden añadir condiciones a las consultas y el orden en el interesa que devuelva los resultados. El siguiente código implementa un método que obtiene informacion de los contactos y la devuelve en un array con el nombre y número de los contactos que tienen número de móvil.

Código Android

```
public ArrayList<String[]> dameContactos() {
        ArrayList<String[]> result = new ArrayList<ContactListItem>();
        ContentResolver cr = getContentResolver();
    // Consulta SQL
    Cursor mCursor = cr.query(
        Data.CONTENT_URI,
        //Columnas a seleccionar
          new String[] { Data.CONTACT_ID, Data.DISPLAY_NAME,
                    Phone.NUMBER,  Phone.TYPE,
                   ContactsContract.CommonDataKinds.Photo.CONTACT_ID},
        //Condiciones
        Phone.NUMBER + " IS NOT NULL",
        null,
        //Orden
        Data.DISPLAY_NAME + " ASC");
    startManagingCursor(mCursor);

    // Iteramos sobre la lista de resultados a través del cursor.
```

```
        int nameIndex = mCursor.getColumnIndexOrThrow(Data.DISPLAY_NAME);
        int numberInde x = mCursor.getColumnIndexOrThrow(Phone.NUMBER);
        int contactoIndex = mCursor.getColumnIndex(Data.CONTACT_ID);

        if (mCursor.moveToFirst()) {
           do {
              String name = mCursor.getString(nameIndex);
              String number = mCursor.getString(numberIndex);
              result.add(new String[] {name, number});
           } while (mCursor.moveToNext());
        }
        return result;
}
```

8. SENSORES

Manuel Báez – Jorge Cordero – Miguel González

Android dispone del paquete ***Android.Hardware*** para dar soporte a la cámara del dispositivo y a otros sensores.

<uses-feature>

Los sensores utilizados por una aplicación deberán de estar declarados en el AndroidManifest.xml haciendo uso de la etiqueta <uses-feature> para que la aplicación pueda hacer uso de ellos. Está información es utilizada por el Android Market para saber si una aplicación es compatible con un dispositivo concreto.

Por ejemplo si la aplicación que queremos desarrollar necesita usar el acelerómetro, incluiremos una linea en el AndroidManifest.xml e indicaremos que es requerido (atributo android:required) como se muestra en el siguiente código.

Código de AndroidManifest.xml

```
...
<uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true" />
<uses-feature android:name="android.hardware.bluetooth" android:required="false"/>
<uses-feature android:name="android.hardware.camera"/>
...
```

El atributo android:required está por defecto a *true*, por lo que si no lo incluimos Android entenderá que el hardware indicado será necesario para la aplicación como sucede con la cámara en el código anterior.

Las clases e interfaces disponibles en android.hardware para el control de sensores, son:

Nombre	Tipo	Descripción
SensorManager	Clase	Gestiona los sensores del dispositivo.
Sensor	Clase	Representa a un sensor
SensorEvent	Clase	Representa el evento de un gestor
SensorEventListener	Interfaz	Recibe los cambios de un sensor.

Para hacer uso de un sensor hay que implementar la clase SensorEventListener y obtener una instancia del SensorManager. Esta instancia la utilizaremos para indicar que queremos registrar los cambios del sensor mediante el metodo registerListener(SensorEventListener listener, Sensor sensor, int rate, Handler handler) del SensorManager. El primer argumento de este método es la instancia de la clase que implementa la interfaz Sensor EventListener. El segundo parámetro es el sensor que queremos registrar. El último parámetro es para indicar la frecuencia ideal con la que nos gustaría registrar el sensor, la cual puede no corresponder con la real.

Una vez indicado al SensorManager que sensor vamos a registrar disponemos de dos métodos para manejar los eventos registrados:

- public void onAccuracyChanged(Sensor sensor, int accuracy)
- public void onSensorChanged(SensorEvent event)

Una vez queramos dejar de registrar los cambios en los sensores debemos llamar a la función unregisterListener(SensorEventListener listener) de la instancia del SensorManager.

El SensorEvent contiene todos los datos en relación al evento.

Tipo	Nombre	Descripción
public Sensor	sensor	Sensor implicado en el evento
public int	accuracy	Precisión del sensor
public long	timestamp	Tiempo (ns) en el que ocurrió el evento
public final float[]	values	Contiene la información del sensor dependiendo del sensor

Método	Valor devuelto
public String getName ()	Nombre del sensor
public float getMaximumRange ()	Valor máximo que alcanza el sensor
public float getPower ()	Potencia (mA) consumida por el sensor cuando está en funcionamiento
public int getMinDelay ()	Tiempo (ns) mínimo que transcurre entre dos eventos de un Sensor
public int getType()	Tipo del sensor

public int getVersion()	Versión del sensor
public float getResolution()	Resolución del sensor

Ejemplo de código:

Código Android

```
public class miActividad extends Activity implements SensorListener {
SensorManager miSensorManager = null;
//... resto de código
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
     //... resto de código

    //Obtenemos la instancia del SensorManager

    miSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
}

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.getType() == Sensor.TYPE_ACCELEROMETER) {
          //Tratamiento del evento si es

    }
    //Tratamiento de eventos de otros sensores
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
protected void onResume() {
    super.onResume();
    //Indicamos al sensor manager que registre los cambios del acelerómetro
    miSensorManager.registerListener(this,

                SensorManager.SENSOR_ACCELEROMETER,
                SensorManager.SENSOR_DELAY_NORMAL);
}
   @Override
protected void onStop() {
    miSensorManager.unregisterListener(thejamos de registrar los cambios del sensor);
    super.onStop();
```

```
} }
```

GESTOS

Android es un sistema operativo para dispositivos móviles, donde el puntero clásico de los sistemas operativos convencionales pasa a ser el contacto táctil del usuario en la pantalla. Podemos ver que algunas aplicaciones tienen funcionalidades distintas cuando un usuario toca la pantalla, incluso variando en función de la duración o forma de dicho toque. Ésto es lo que se llama un gesto en Android y vamos a ver cómo implementarlos.

Estos gestos dan a Android un gran potencial a la hora de poder realizar un gran abanico de acciones con un escaso número de gestos, hay aplicaciones que su única funcionalidad consiste en ahorrarnos tiempo haciendo que ciertos gestos se conviertan en una especie de accesos directos. Incluso hay un teclado llamado Swype que es un teclado mediante gestos haciendo que sea mucho más ágil que un teclado tradicional.

Hay varios tipos de gestos que podemos capturar, para ello tendremos que utlizar GestureDetector.OnGestureListener que nos notificará cuando salta un evento dentro del listener. Tendremos a nuestra disposición los siguientes subtipos del listener:

Métodos:

abstract boolean	onDown(MotionEvent e)	Notifica cuando se produce un down devolviéndolo en el MotionEvent e.
abstract boolean	onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)	Notifica una pulsación mantenida devolviendo el primer punto en MotionEvent e1 y el final en MotionEvent e2.
abstract void	onLongPress(MotionEvent e)	Notifica cuando se produce una pulsación larga devolviendo los datos en MotionEvent e.
abstract boolean	onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)	Notifica cuando se está moviendo un scroll edvolviendo los valores de la posición inicial en MotionEvent e1 y la final en MotionEvent e2
abstract void	onShowPress(MotionEvent e)	Indica que se ha seleccionado un elemento pero no se ha dejado de pulsar todavía, para mostar al usuario se ha reconocido su interacción.

abstract boolean	onSingleTapUp(MotionEvent e)	Notifica cuando se produce un up devolviéndolo en MotionEvent e

A continuación vamos a ver un ejemplo con el onFling haciendo el gesto de izquierda a derecha:

Código Android

```
public class ContactosActivity extends Activity {
    private static final int SWIPE_MIN_DISTANCE = 120;
    private static final int SWIPE_MAX_OFF_PATH = 250;
    private static final int SWIPE_THRESHOLD_VELOCITY = 200;
    private ContactArrayAdapter listAdapter = null;
    private GestureDetector gestureDetector = null;
    private ContactosActivity actividad = null;
    //private ProgressDialog progressDialog = null;
    class MyGestureDetector extends SimpleOnGestureListener {
      @Override
      public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
                    float velocityY) {
              Intent intent = new Intent(ContactosActivity.this, GruposActivity.class);
              if (Math.abs(e1.getY() - e2.getY()) > SWIPE_MAX_OFF_PATH) {
                    return false;
              }
              // right to left swipe
              else if (e1.getX() - e2.getX() > SWIPE_MIN_DISTANCE
                          && Math.abs(velocityX) >
SWIPE_THRESHOLD_VELOCITY) {
  // Tu código para cuando hagas el gesto de izquierda a derecha
              }
              return false;
      }
      // Es necesrio devolver true en el onDown para que el evento onFling lo registre
      @Override
```

```
        public boolean onDown(MotionEvent e) {

                return true;

        }}}
```

9. MULTIMEDIA

Daniel Sanz – Mariam Saucedo – Pilar Torralbo

MULTIMEDIA EN ANDROID

Hoy en día, los dispositivos móviles no solo se usan para hablar y mandar mensajes de texto. La gran mayoría de los mismos permiten, grabar y reproducir audio y video, así como realizar fotografías. En este tema, se intenta presentar todas las posibilidades mencionadas anteriormente, proporcionadas por Android, así como dar a conocer algunos ejemplos de implementación de pequeñas funciones para reproducir audio o hacer fotografías.

El Framework[1] proporcionado por Android para multimedia, permite capturar y reproducir audio, vídeo o imágenes en los formatos más comúnmente usados, pudiendo interactuar con archivos contenidos en el propio terminal Android, con archivos externos al dispositivo, o con aquellos dispuestos a través de Internet. Para la reproducción utiliza MediaPlayer o JetPlayer, siendo MediaRecorder el usado para grabar audio o vídeo, o hacer fotografías a través de la cámara del terminal. Android soporta varios formatos de audio, video e imágenes. Algunas de ellas como mp3, midi y flac para audio; 3gp y mpeg para videos; y jpg y png entre otros para imágenes.

REPRODUCCIÓN DE AUDIO

En la reproducción de audio, Android proporciona sus propias clases destinados a ello. A continuación, se describirá como usar la clase "**MediaPlayer**" para programar en nuestra propia aplicación y así crear un reproductor propio.

Para poder usar la clase "MediaPlayer", primero hay que especificar los permisos necesarios para que no dé problemas a la hora de compilar el programa. Para ello, en el AndroidManifest se incluye lo siguiente:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Esto permitirá la conexión a Internet en el caso de que se reproduzca audio o video extraído de la web. Para llevar un mejor manejo de la reproducción, se puede incluir un par de botones como son "start" y "pause"

(1) Framework es un término que significa "marco". Es decir, proporciona un conjunto de funciones para implementar aplicaciones con características semejantes.

Si además se quiere poder reproducir multimedia con el dispositivo en suspensión o bloqueado (usando los métodos setScreenOnWhilePlaying() o setWakeMode() que más adelante serán explicados), se añadirá la línea:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Como ya se ha indicado antes, MediaPlayer permite reproducir audio o video desde diferente puntos:

- Recursos locales (los cuales se encontrarán guardados en la carpeta "raw" del proyecto).
- URI interna (Identificador Uniforme de Recurso en inglés, ya sea guardado en el dispositivo o en una tarjeta SD).
- URL externa (obtenida de una página web).

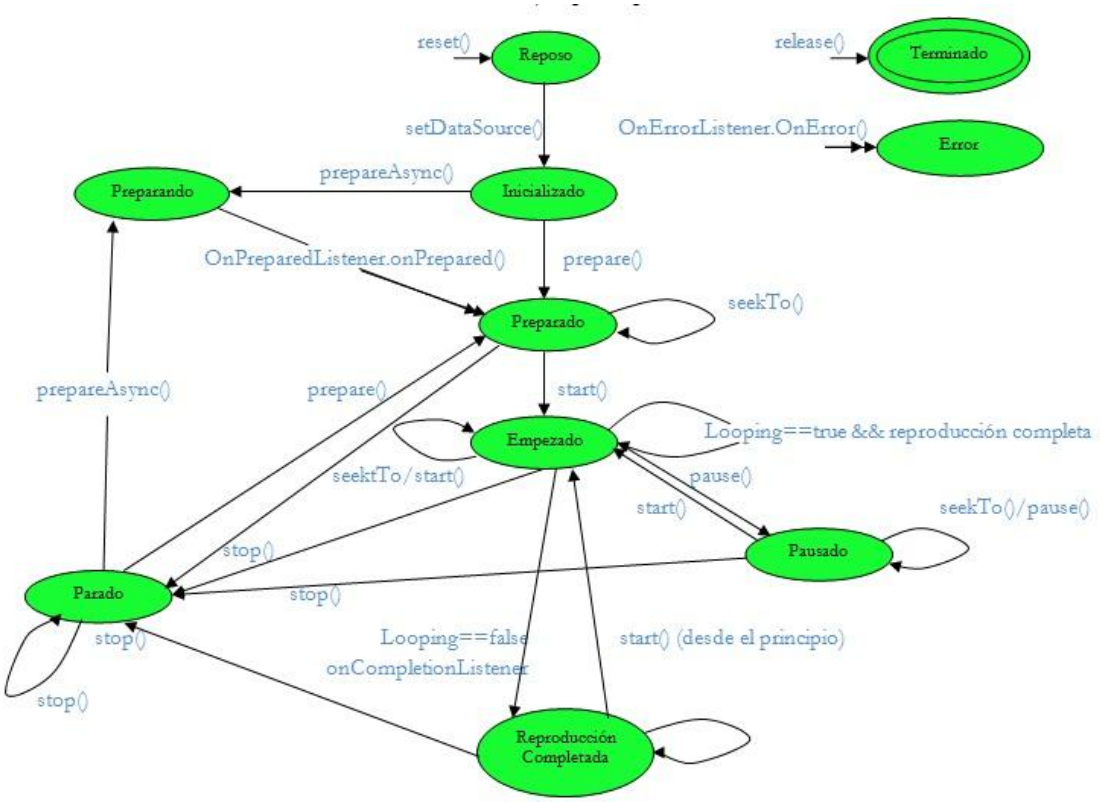


Figura 1. Funcionamiento de la clase MediaPlayer

El funcionamiento de esta clase funciona como una máquina de estados. Cuando el objeto MediaPlayer es creado mediante new() o después de llamar a la función release(), se dice que está en un estado de reposo. Hasta llegar a este estado mediante una de las dos llamadas, el objeto pasará por todo su ciclo de vida.

El ciclo de vida de una reproducción sin tener errores pasaría por los siguientes estados: **Reposo** (cuando se crea el objeto MediaPlayer), **Inicializado** (se le indica al objeto cuál va a ser la pista o video a reproducir), **Preparado** (el MediaPlayer ya tiene todo listo para reproducir), **Empezado** (MediaPlayer está reproduciendo), **Parado** (la pista terminó y MediaPlayer espera una nueva instrucción).

Para poder usar un objeto MediaPlayer, se invoca la constructora por defecto que proporciona Android. Una vez construido el objeto, se indica desde donde va a reproducir la pista. En el caso de que se quiera reproducir desde la carpeta raw del proyecto, hay que especificarle el lugar (en este caso la carpeta raw) y el nombre del archivo que se quiera reproducir:

```
MediaPlayer mp = new MediaPlayer();
mp = MediaPlayer.create(R.raw.cancion_prueba.mp3);
mp.start();
```

Si el archivo que se va a usar se encuentra en la memoria del dispositivo o en una tarjeta SD, mediante una variable de tipo URI se indica la ruta del teléfono en la cual se encuentra el archivo a reproducir:

```
Uri miUri = .... ;//Inicializa la variable apuntando al archivo deseado
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(getApplicationContext(), miUri);
mp.prepare();
mp.start();
```

Si por el contrario el archivo se desea reproducir desde una url, mediante una conexión http indicamos al objeto desde donde descargarlo (hay que tener en cuenta que primero debe descargar el archivo, por lo que puede tardar un tiempo dependiendo de la velocidad de conexión):

```
String url = "http://...";//Se introduce la url desde la que operar
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(url);
mp.prepare();
mp.start();
```

Como se ha podido comprobar, tanto para reproducir desde una URI interna como desde una url externa, es necesario preparar el MediaPlayer para obtener el archivo, al contrario que con un archivo local que se encuentra dentro del proyecto y listo para usarse. En el caso de querer manejar también el audio de la reproducción, se incluirá la clase "AudioManager" la cual primero hay que escribir los permisos necesarios para poder utilizarla:

```
<accion android:name="android.media.AUDIO_BECOMING_NOISY" />
```

Y así poder obtener y manejar a nuestro antojo el volumen de la aplicación. En el siguiente ejemplo se puede observar la forma de obtener el volumen máximo permitido por la aplicación y el volumen actual:

```
AudioManager aManager = (AudioManager) getSystemService(AUDIO_SERVICE);
                //Captura las especificaciones del volumen de la aplicación
float actualVolumen = (float) aManager.getStreamVolume(aManager.STREAM_MUSIC);
float maxVolumen = (float) aManager.getStreamMaxVolume(aManager.STREAM_MUSIC);
```

Una vez obtenidos estos datos, se podrá manejar el volumen de forma sencilla.

REPRODUCCIÓN DE VIDEO

Para programar un reproductor de video en nuestra aplicación, será necesario incluir en la interfaz (archivo .xml) un control de "VideoView" que se encuentra en la paleta de "Images & Media". Para llevar un mejor manejo de la reproducción del video, además podemos añadir un par de botones para reproducir y pausar el video.

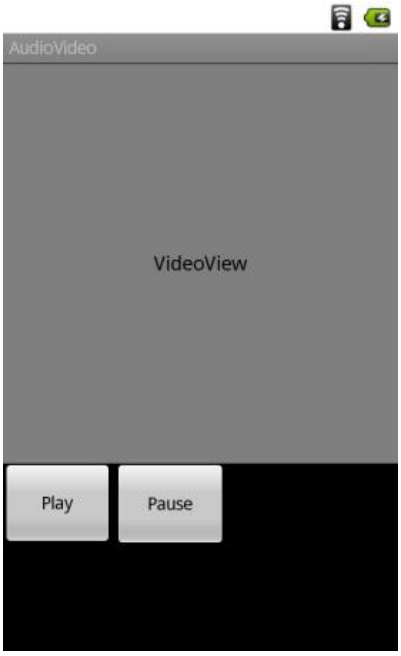


Figura 2. Reproducción de video

En este caso, podremos reproducir un video que esté alojado en la tarjeta SD de nuestro dispositivo (URI) o a través de una URL externa de la web. Para ello, hay que diferenciar los dos métodos de linkado para reproducir uno u otro video.

Para la reproducción a través de un objeto de tipo URI, escribimos las siguientes instrucciones:

```
Uri uri = ....;  //Se incluye la ruta del teléfono donde se encuentre el video
VideoView video = ..... //linkado con su id correspondiente al archivo .xml
video.setVideoUri(uri);
```

Y en el caso de que reproduzca a través de una url externa:

```
String url = ....;  //URL desde donde se descargará el video
VideoView video = .....;
video.setVideoPath(url);
```

Para terminar, proporcionamos a los dos botones un setOnClickListener() para que cada uno realice su función. En el caso del botón play, la instrucción a escribir será video.play(). Y en el caso del botón pause la instrucción será video.pause().

GRABAR SONIDO

El framework multimedia de Android incluye un soporte que permite capturar y codificar una gran variedad de formatos comunes de audio, de modo que estos se puedan integrar fácilmente en la aplicación. Para ello, se puede usar la API "MediaRecorder" siempre y cuando sea compatible con el hardware del dispositivo (que tenga micrófono o no). Sin embargo, el emulador de Android no posee la capacidad de capturar audio, por lo que toda prueba que se precie se deberá hacer en un dispositivo que disponga de micrófono.

Antes de nada, se deberán incluir los permisos necesarios para poder escribir desde una entrada externa en la tarjeta SD y para poder grabar audio. En el archivo AndroidManifest se escribirá:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

La captura de audio desde el dispositivo es más complicado que la reproducción de video o audio. Aún así, se puede programar de forma sencilla. En primer lugar, hay que crear un objeto de tipo MediaRecorder. Del mismo modo se puede crear otro objeto MediaPlayer (visto en este tema), para reproducir la pista capturada:

```
MediaRecorder mRecorder = new MediaRecorder();
MediaPlayer mPlayer = new MediaPlayer();
```

A continuación, se definirá el micrófono como medio para capturar el audio y se define que el archivo se va a guardar con la especificación 3GPP, con extensión .3gp y el codec que se va a emplear:

```
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

Una vez en este punto, hay que indicar la dirección de la tarjeta SD; indicándole al objeto MediaRecorder que, una vez realizada la captura de audio, debe almacenarse en esta dirección. A su vez, creamos un archivo temporal 3gp almacenado en la tarjeta SD donde se guardará el audio:

```
File direccion= new File(Environment.getExternalStorageDirectory().getPath());
File archivo = File.createTempFile("temporal", ".3gp", direccion);
mRecorder.setOutPutFile(archivo.getAbsolutePath());
```

Ahora, ya podemos preparar la captura y empezar con ella. Y pararla una vez se haya terminado de grabar lo deseado:

```
mRecorder.prepare();
mRecorder.start();
.....
mRecorder.stop();
mRecorder.release();
```

Al igual que se ha visto en la reproducción de video, se puede incluir en la interfaz unos botones para tener un mejor control del inicio, parado y reproducción de la grabación de audio.

GRABAR VIDEO

Al igual que pasaba en el apartado anterior, para la captura de video se necesita un dispositivo que disponga de una cámara. También se usará un objeto de tipo "MediaRecorder" solo que esta vez le indicaremos que la captura se obtendrá de la cámara. En este caso, a parte de los permisos que se declararon para la grabación de audio, hay que añadirle los permisos para grabar video y para usar la cámara:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.RECORD_VIDEO" />
<uses-permission android:name="android.permission.CAMERA" />
```

Figura 3. Grabación de video

En la interfaz, hay que añadir un "SurfaceView", el cual permitirá tener un espacio dedicado a dibujar frames del vídeo para la vista y la reproducción (en la activity principal hay que implementar la interfaz "SurfaceHolder.Callback"). Para este apartado, servirá para la vista previa de la grabación y para reproducirla posteriormente. Además, se podrá añadir tres botones que implementen las funciones grabar, detener y reproducir del "SurfaceView". Como consejo, se puede configurar la rotación de la pantalla para que se quede bloqueada y no re-cree la activity al girar el dispositivo. Para se escribe en el manifiesto la siguiente línea:

```
android:screenOrientation="portrait"
```

En esta aplicación se definirán cuatro variables importantes, el "MediaPlayer" y "MediaRecorder" vistos anteriormente y una variable con el nombre del archivo donde se va a guardar la grabación.

De los métodos heredados de la interfaz, importan dos de ellas. El método "surfaceCreated", en el cual se inicializaran las variables y se propone el "SurfaceHolder" como display para la grabación y la reproducción. Y el método "surfaceDestroyed", en el que se liberarán las variables usando el método release() de éstas.

```
mRecorder.setPreviewDisplay(holder.getSurface());
mPlayer.setDisplay(holder);
```

Ahora se necesitará agregar la cámara para que el objeto "MediaRecorder" use ésta para la grabación. Se configura las fuentes de audio y video. Y se llama a los métodos unlock() y lock() de la cámara. Esto sirve para que ninguna otra aplicación pueda utilizar la cámara mientras esté la nuestra en funcionamiento. El primero se llamará cuando se quiera usar la cámara, y el segundo cuando se haya terminado.

```
mCamara = getCameraInstance();
mCamara.unlock();
mRecoder.setCamera(mCamara);
mRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
mRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
mRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toSting());
```

Una vez llegados a este punto, ya se pueden configurar los botones para el uso de la grabación, usando los comandos start() y stop() del "MediaRecorder" para tener manejo sobre la grabación del video, los mismos comandos para el "MediaPlayer" para la previsualización de la grabación y, una vez terminado el uso de la cámara para grabar el video, el método lock() para bloquearla de nuevo.

10. CREACIÓN DE UN WIDGET

Álvaro Borrego – Francisco Hernández – David Palomero

CREACIÓN DE UN WIDGET SENCILLO

En esta sección del manual se va a explicar cómo crear un widget para Android. En primer lugar se mostrará y explicará con detalle la manera de crear un widget estático, es decir, sin ninguna funcionalidad. De esta manera se pretende que se consiga entender de manera sencilla y clara la estructura de este tipo de componentes muy utilizado en cualquier aplicación Android.

Una vez entendida la estructura de un widget, se añadirá una funcionalidad más avanzada, como puede ser, la realización de una determinada acción al ser pulsado por el usuario o su actualización automática.

El widget que se va a implementar en esta primera parte consistirá en una imagen de un reloj y un texto con un mensaje ('Hora Actual'). Con este ejemplo se pretende comprender de una manera sencilla la estructura de un widget en Android, sin tener en cuenta aspectos más avanzados, como puede ser la interacción con el usuario.

DEFINICIÓN DE LA INTERFAZ GRÁFICA

Como se ha comentado anteriormente, el widget consistirá en una imagen de un reloj y un mensaje de texto, que en un principio, solamente mostrará el mensaje 'Hora Actual'. La imagen general del widget en esta primera parte será:

Figura 1. Creación de un widget

Para conseguir este aspecto, se creará en primer lugar, un layout xml llamado *widgethora.xml*. Éste layout está formado sencillamente por un contenedor *LinearLayout* en el que se sitúan una imagen *ImageView* y una etiqueta de texto *TextView* que muestra el mensaje.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
      android:padding="5dip">

            <ImageView
                  android:layout_width="wrap_content"
                  android:src="@drawable/reloj"
                  android:layout_height="wrap_content"
                  android:layout_gravity="right"
                  android:id="@+id/imageView1"/>

            <TextView android:id="@+id/textViewHora"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:textColor="#000000"
                  android:text="Hora Actual" />
</LinearLayout>
```

NOTA: En el ejemplo se ha utilizado una imagen llamada reloj.png, por lo que es necesario crear una imagen con ese nombre e incluirla en el proyecto. Otra opción posible, es sustituir android:src="*@drawable/reloj*" por android:src="*@drawable/icon*", de esta manera el icono del widget no será el de un reloj sino el icono por defecto de Android.

ASIGNAR PROPIEDADES AL WIDGET

En esta parte se da la posibilidad de definir algunas de las propiedades que va a tener el widget, como por ejemplo, el tamaño en pantalla, el nombre que aparecerá en el menú etc... Para ello, se crea un nuevo xml (que se ubicará en la carpeta \res\xml del proyecto) llamado *widgethora_provider.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:initialLayout="@layout/widgethora"
      android:label="Widget de la Hora"
      android:minWidth="146dip"
      android:minHeight="72dip"/>
```

En este caso se han definido las siguientes propiedades:

- **initialLayout:** hace referencia al layuout xml creado anteriormente, en este caso, *widgethora*.
- **label:** será el nombre que aparecerá en el menú de aplicaciones de Android.
- **minWidth/minHeight:** define el ancho y el alto respectivamente del widget en pantalla. En Android, la pantalla se divide en pequeñas celdas donde se pueden colocar iconos de aplicaciones, iconos de contactos, widgets, etc.

En el ejemplo, se quiere tener un widget con unas dimensiones de dos celdas de ancho y una celda de alto (2x1). Para conseguir estas dimensiones, existe una fórmula que indica el valor exacto que deben tener las propiedades minWidth y minHeight:

dimensión = (número de celdas * 74) - 2

Teniendo en cuenta esta fórmula, y sustituyendo el número de celdas deseadas para cada dimensión, se obtienen los valores minWidth="146dip" y minHeight="72dip". Existen otras propiedades muy interesantes que no se han tenido en cuenta en este ejemplo, pero que serían de gran utilidad en la mayoría de aplicaciones con widgets, como pueden ser:

- **icon:** icono que se muestra para el widget en el selector de AppWidget.
- **minResizeWidth/minResizeHeight:** anchura/altura mínima que se puede cambiar de tamaño para el widget.
- **resizeMode:** Las reglas por las que se puede cambiar el tamaño de un widget.
- **updatePeriodMillis**: frecuencia en la que se actualizará el widget, definida en milisegundos.

IMPLEMENTACIÓN DE LA CLASE PRINCIPAL

En este paso, se declarará la clase principal (que será la que se mostrará al pulsar sobre el icono en el menú). En este ejemplo, se mostrará una pantalla en negro con un mensaje que mostrará la forma de añadir el widget creado al escritorio.

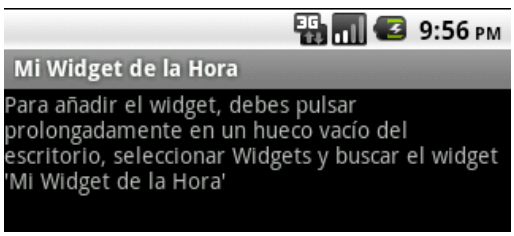


Figura 2. Información de un widget

El código de esta clase es el siguiente:

```
public class WidgetHoraMain extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

El layout *main* estará formado únicamente por una etiqueta de texto, explicando la forma de añadir el widget al escritorio.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/mensaje"/>

</LinearLayout>
```

Se debe incluir en el archivo *\res\values\strings.xml* la siguiente entrada:

```
<string name="mensaje">"Para añadir el widget, debes pulsar
      prolongadamente en un hueco vacío del escritorio, seleccionar
      Widgets y buscar el widget 'Mi Widget de la Hora'"
</string>
```

Como se ha comentado en repetidas ocasiones a lo largo de la explicación, en esta primera parte del ejemplo no se va a implementar ninguna funcionalidad del widget, no obstante, se va a declarar y a explicar brevemente la clase necesaria para esta tal efecto ya que será implementada más adelante. Es necesario crear la clase *MiWidgetHora.java:*

```
public class MiWidgetHora extends AppWidgetProvider {
      public void onUpdate(Context context,
                           AppWidgetManager appWidgetManager,
                           int[] appWidgetIds) {
      }

}
```

Los principales eventos en un widget son:

- **onEnabled():** lanzado cuando se añade al escritorio la primera instancia de un widget.
- **onUpdate():** lanzado periódicamente cada vez que se debe actualizar un widget.
- **onDeleted():** lanzado cuando se elimina del escritorio una instancia de un widget.
- **onDisabled():** lanzado cuando se elimina del escritorio la última instancia de un widget.

AÑADIR EL WIDGET AL MANIFEST DEL PROYECTO

Por último, queda declarar el widget en el archivo *manifest* del proyecto.

```
<receiver android:name=".MiWidgetHora" android:label="Mi Widget de la
Hora">
<intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
</intent-filter>
<meta-data
    android:name="android.appwidget.provider"
    android:resource="@xml/widgethora_provider" />
</receiver>
```

WIDGET CON FUNCIONALIDAD

Una vez explicadas las características más importantes para la creación de un widget, llega el momento de añadirle funcionalidad, permitiendo su actualización e interacción con el usuario. En esta segunda parte, se va a utilizar el widget creado anteriormente, añadiendo una serie de eventos que permitan realizar una acción concreta al ser pulsado por el usuario.

La idea de este widget más avanzado, será mostrar la hora actual del dispositivo al ser pulsado por el usuario. También se añadirá la opción de actualizarse automáticamente pasado un periodo de tiempo determinado. El layout *widgethora.xml* no se ha modificado, por lo que el widget seguirá teniendo la misma apariencia. El objetivo, es ahora, mostrar la hora actual del dispositivo en lugar de un mensaje estático.

Figura 3. Widget más avanzado

DEFINIR EL TIEMPO DE ACTUALIZACIÓN DEL WIDGET

Como una de las funciones del widget será la de actualizarse automáticamente, habrá que definir el tiempo de actualización (en milisegundos). Para ello, simplemente hay que añadir la siguiente propiedad en el widgethora_provider.xml:

```
android:updatePeriodMillis="1800000"
```

Una opción muy interesante, pero que no se implementará en este caso debido a la simplicidad del ejemplo, es la de mostrar una pantalla de configuración del widget, de esta manera el usuario podrá seleccionar las características que le interesen. Para mostrar una pantalla de configuración, bastaría con añadir, a este mismo archivo:

```
android:configure="rutaDelPaquete.ConfigWidget"
```

Siendo ConfigWidget la actividad donde se definen las distintas opciones de configuración. Con esto, al colocar el widget en el dispositivo, se ejecutará automáticamente la pantalla de configuración antes de ser colocado.

AÑADIR FUNCIONALIDAD AL WIDGET

Una vez realizadas estas pequeñas modificaciones, es el momento de implementar la funcionalidad. En la primera parte, se creó una clase llamada MiWidgetHora.java que quedó prácticamente vacía (ya que no se deseaba implementar ninguna funcionalidad). Ahora es necesario completar esa clase, ya que va a ser la encargada de realizar todas las acciones relacionadas con el widget.

EVENTO ON_UPDATE()

El evento onUpdate (lanzado periódicamente cada vez que se debe actualizar un widget) recibe como parámetro una lista de todas las instancias añadidas al escritorio. Cuando se actualiza, es necesario actualizar todas estas instancias. Para ello, se recorre dicha lista y se van actualizando una a una llamando al método actualizar, que será explicado más adelante.

```
public void onUpdate(Context context,AppWidgetManager
                     appWidgetManager,int[] appWidgetIds) {

      int i=0;
      while (i<appWidgetIds.length)
      {
            int id=appWidgetIds[i];
            actualizar(context, appWidgetManager, id);
            i++;
      }
}
```

EVENTO ON_RECEIVE()

El evento *onReceive*, es el encargado de capturar los mensajes enviados por las componentes. Dependiendo del tipo de mensaje recibido, se realizará una función u otra. En este ejemplo sólo se va a capturar un tipo de mensaje, que será el enviado al pulsar sobre el icono del reloj de la interfaz. Para indicar esto, se creará al principio de la clase un atributo que hará referencia a este tipo de mensajes.

```
private static final String ACCION_PULSAR="PULSAR";
```

El primer paso a realizar en éste método, es capturar la acción asociada al *intent*:

```
final String action = intent.getAction();
```

Una vez que se tiene esta acción, es necesario saber de qué tipo es. Come se ha comentado anteriormente, en este ejemplo, sólo se tiene un tipo de mensaje (ACCION_PULSAR), por lo que bastará con comprobar si el mensaje capturado es de este tipo. En caso de ser así, se debe conocer la id del widget pulsado, ya que es posible tener más de un widget de este mismo tipo en pantalla. Finalmente, sólo queda llamar al método actualizar, encargado de realizar la acción sobre el widget pulsado. Dicho método, necesita como uno de sus argumentos un widget manager, por lo que es necesario obtenerlo antes de realizar la llamada:

```
AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);
```

El código completo de *onReceive* se muestra a continuación:

```
public void onReceive(Context context, Intent intent) {
  final String action = intent.getAction();

  if (action.equals(ACCION_PULSAR)){
      final int id = intent.getIntExtra
                        (AppWidgetManager.EXTRA_APPWIDGET_ID,
                         AppWidgetManager.INVALID_APPWIDGET_ID);

      AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);
      actualizar(context, widgetManager, id);

  }
  super.onReceive(context, intent);
}
```

MÉTODO ACTUALIZAR

El método actualizar, es el encargado de realizar la actualización del widget. Un widget está formado por una serie de componentes llamadas Remote Views. Para hacer referencia a cada una de estas vistas, es necesario acceder a la lista de componentes:

```
RemoteViews componentes = new RemoteViews(context.getPackageName(),
                                    R.layout.widgethora);
```

El objeto componentes, contiene una lista con todas las vistas que forman el widget, en este caso, un *TextView* y un *ImageView*. La forma de hacer referencia a cada una de ellas es la siguiente:

```
componentes.setTextViewText(R.id.textViewHora,"...");
```

Con esto, se consigue asignar al elemento de tipo *TextView* con id *textViewHora* (definido en el *layaout widgethora.xml*) el string encerrado entre comillas. Ahora, toca el turno de hacer lo mismo con el componente *imageviewReloj* (la imagen del reloj). En este caso, lo que se desea es definir el evento *OnClick* sobre la imagen, para así poder actualizar el widget cuando es pulsada por el usuario. Como se ha comentado anteriormente en el *OnReceive*, la forma de conseguir asignar un evento de tipo *OnClick* a una componente de un widget, es enviando un mensaje de tipo ACCION_PULSAR. Para ello, lo primero que hay que hacer es crear un ***intent*** y asociarle la acción comentada. También será necesario conocer el id del widget pulsado.

```
final Intent onClickIntent = new Intent(context, MiWidgetHora.class);
onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);
```

Como la finalidad de este widget es la de mostrar la hora actual del dispositivo, va a ser necesario implementar un método que realice dicha tarea:

```
private static String dameHora(){
      Calendar calendario = new GregorianCalendar();
      return calendario.getTime().toLocaleString();
}
```

Para asegurar la correcta actualización de los componentes del widget en la interfaz, es necesario añadir al final de este método, la siguiente línea:

```
appWidgetManager.updateAppWidget(id, componentes);
```

De esta manera se consigue refrescar de manera adecuada todos los componentes del widget.

El código completo del método *actualizar* queda de la siguiente manera:

```
public static void actualizar(Context context, AppWidgetManager
                              appWidgetManager, int id){

    RemoteViews componentes = new RemoteViews(context.getPackageName()
                                              ,R.layout.widgethora);

    final Intent onClickIntent = new Intent(context,
                                            MiWidgetHora.class);

    onClickIntent.setAction(MiWidgetHora.ACCION_PULSAR);
    onClickIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id);

    final PendingIntent onclickPendingIntent =
            PendingIntent.getBroadcast(context, id, onClickIntent,
            PendingIntent.FLAG_UPDATE_CURRENT);

    componentes.setOnClickPendingIntent(R.id.imageViewReloj,
                                        onclickPendingIntent);

    String horaActual= dameHora();
    componentes.setTextViewText(R.id.textViewHora, "...");
    appWidgetManager.updateAppWidget(id, componentes);
}
```

11. PUBLICANDO EN EL MARKET

Álvaro Borrego – Francisco Hernández – David Palomero

EL ANDROID MARKET

El Android Market es la tienda de aplicaciones de Android. Tiene un acceso rápido y ágil a aplicaciones creadas por desarrolladores de todo el mundo. Desde el punto de vista del desarrollador, se puede usar el Market para publicar sus propias aplicaciones. Tiene las siguientes características destacables:

- Es libre, cualquier usuario puede apuntarse.
- Las aplicaciones publicadas pueden ser puntuadas y comentadas por los usuarios.
- Cuenta con estadísticas de cada aplicación, tales como número de descargas, puntuaciones, comentarios…
- Las aplicaciones publicadas en él, pueden ser de pago, gratuitas, gratuitas con publicidad o gratuitas con limitaciones. Éstas últimas cuentan con su versión completa de pago.

Una de las grandes ventajas que ofrece el Android Market es la posibilidad de instalar una aplicación desde un PC teniendo el dispositivo conectado.

ACCEDER AL ANDROID MARKET

Existen dos maneras por las que se puede acceder al Market para descargar aplicaciones a los dispositivos: mediante la aplicación preinstalada en los dispositivos Android, señalizada con el icono mostrado en la Figura 1., o mediante la Web oficial https://market.android.com cuya imagen inicial se muestra en al Figura 2.

Figura 1. Icono del Market

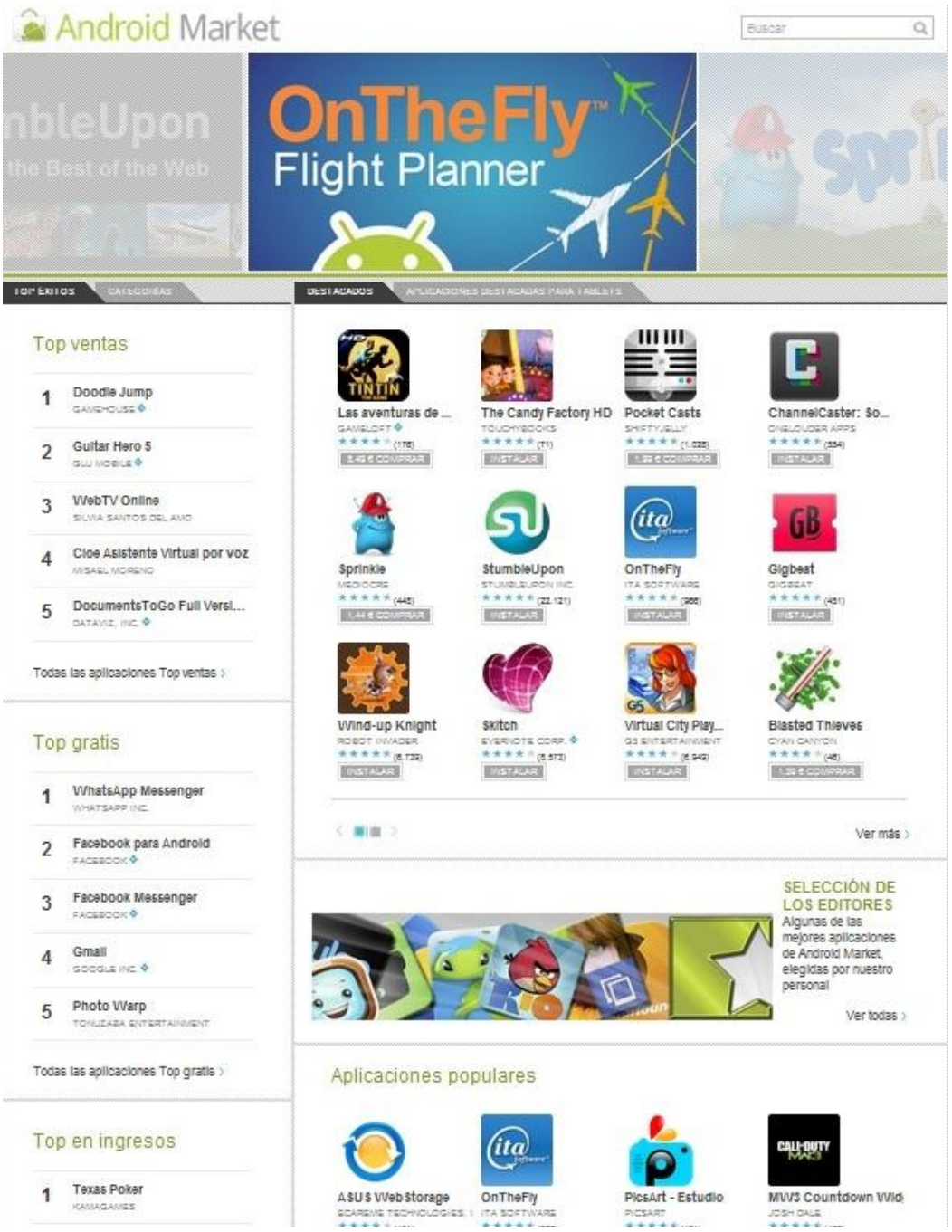


Figura 2. Web oficial de Android Market

En ambas, hay diferentes opciones y las aplicaciones están estructuradas por categorías.La primera opción que presenta es la búsqueda, mediante la cual se puede buscar la aplicación deseada. Una vez introducido el nombre, se mostrarán todas las aplicaciones encontradas, y se podrán clasificar según relevancia, popularidad, precio, etc. A continuación se muestran algunas de las aplicaciones clasificadas por categorias, destacados o aplicaciones top según ventas, nuevas, etc.

PUBLICAR EN ANDROID MARKET

REQUISITOS

Para publicar una aplicación en el Android Market, primero hay que registrarse con una cuenta de desarrollador de Google (Google Checkout 25 $) de acuerdo con los términos del servicio. Se puede registrar como desarrollador en:

http://market.android.com/publish

Una vez registrado, se puede subir la aplicación, actualizar tantas veces como quiera, y publicarla cuando esté lista. Es necesario residir en uno de los países soportados: Australia, Austria, República Checa, Francia, Alemania, Italia, Países bajos, Polonia, Singapur, España, Reino Unido, Estados Unidos. Los requisitos impuestos por el servidor de Android Market son los siguientes:

1. La aplicación debe tener definidos dos atributos en el archivo <manifest>:

   Android:versionCode y android:versionName

2. El atributo versionCode es un número entero creciente que lo utiliza el servidor para identificar internamente la versión de la aplicación y para el manejo de cambios.

3. El atributo versionName muestra a los usuarios la versión de la aplicación. Consiste en una cadena de texto que representa la versión tal y como la verán los usuarios.

4. Además, hay que definir otros dos atributos, android:icon y android:label, dentro del elemento <application> del archivo *manifest*.

5. La aplicación debe estar firmada con una clave criptografica privada

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.prueba"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <activity android:name=".pruebaActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Figura 3. Requisitos de configuración en Manifiest

PASOS

Se publica la aplicación a través de la herramienta web disponible en el portal de desarrolladores del Android market, cumplimentando los datos necesarios en tres partes:

1. La parte de "**subir recursos**", donde se sube el .apk, las imágenes promocionales, un icono, un gráfico promocional, y un video de youtube (Figura 4).
2. La parte de "**especificación de detalles**" con la descripción de la aplicación en los diferentes idiomas, tipo, categoría, descripción (Figura 5).
3. Y la parte de '**opciones de publicación**' con la protección y clasificación por edades, además de la información de contacto y aceptaciones correspondientes (Figuras 6 y 7).

Una vez publicada la aplicación (opción publish), en breves minutos ya se encontrará en el Market.

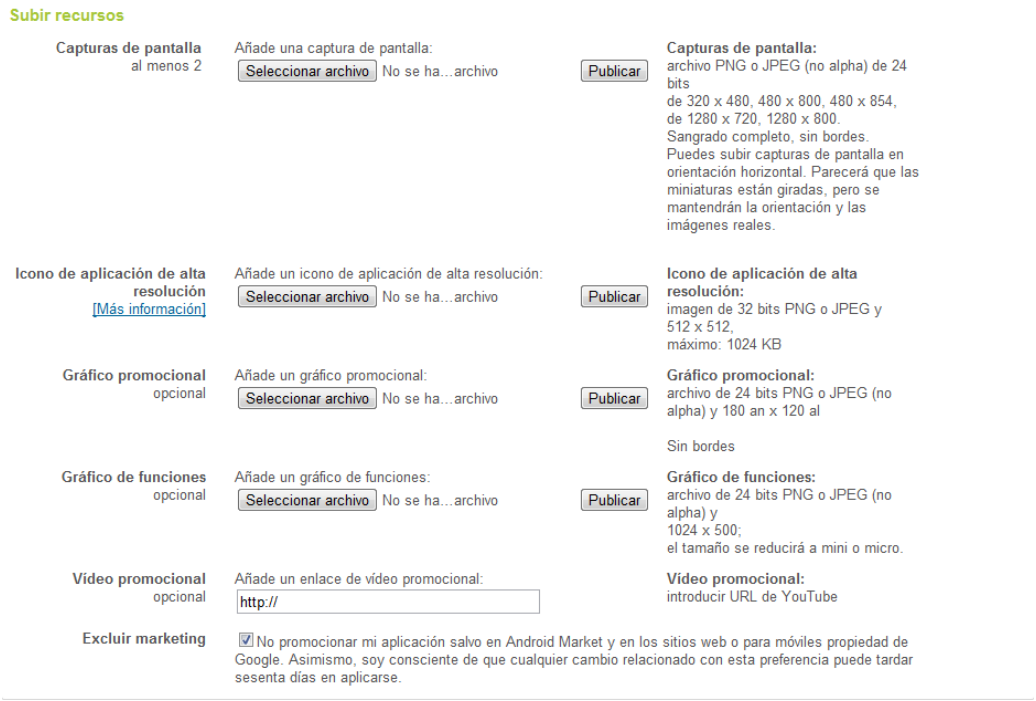


Figura 4. Subir recursos

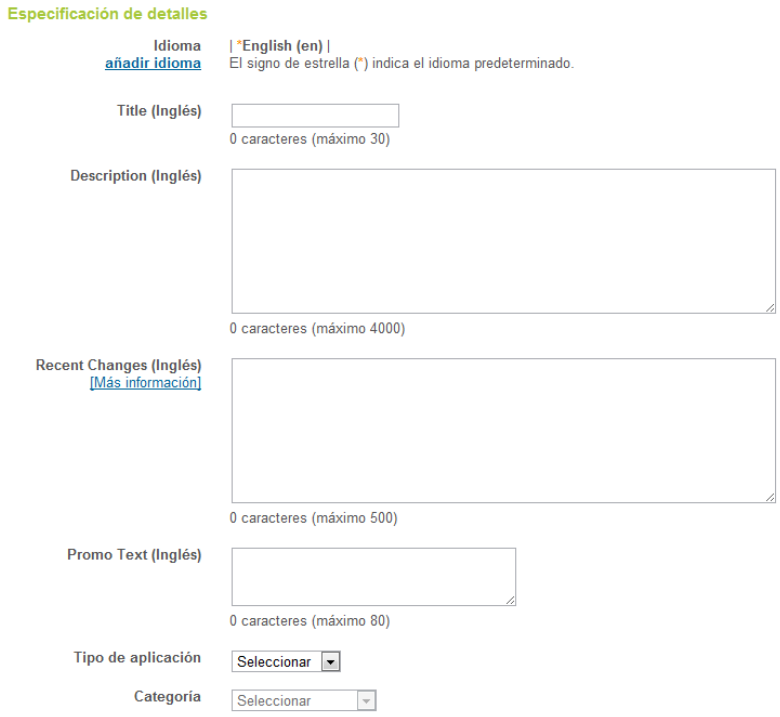


Figura 5. Especificación de detalles

Figura 6. Opciones de publicación

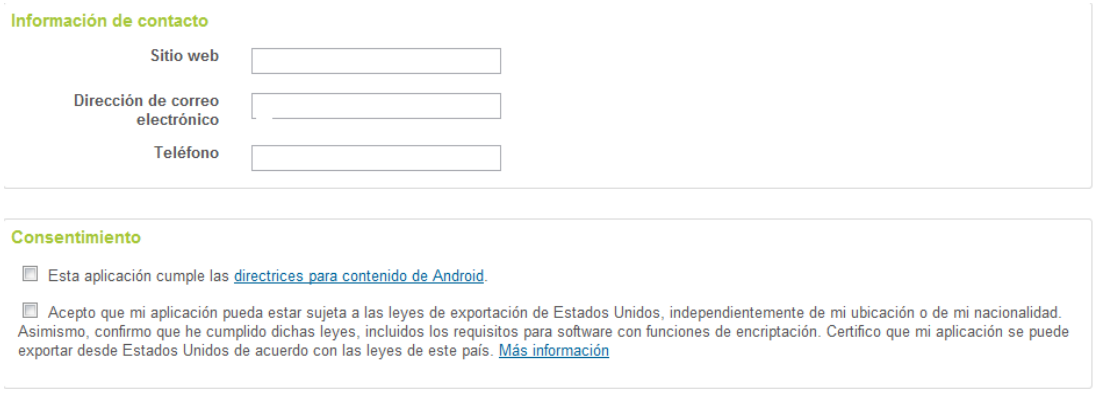


Figura 7. Almacenamiento de información de contacto

FIRMA DIGITAL

Las aplicaciones deben estar firmadas digitalmente para poder distribuirse. Esto es una medida de seguridad, así solo el desarrollador de la aplicación puede modificarla y actualizarla. El certificador puede ser autofirmado, no es necesaria una autoridad certificadora. Existen dos modos para firmar las aplicaciones:

- Debug: mientras desarrollamos. El plug-in ADT se encarga de la firma automáticamente, gestionando claves generadas.
- Release: Crea una clave proporcionando el usuario un password. Para esto se puede utilizar el ADT export wizard desde eclipse.

Firmar aplicaciones desde eclipse a través de export wizard:

Se abre la aplicación que se quiere firmar con eclipse, y en el árbol de directorios se abre el archivo AndroidManifest. Una vez abierto, se abre la primera pestaña llamada *Manifest* y en la sección Exporting hay dos opciones para firmar la aplicación.

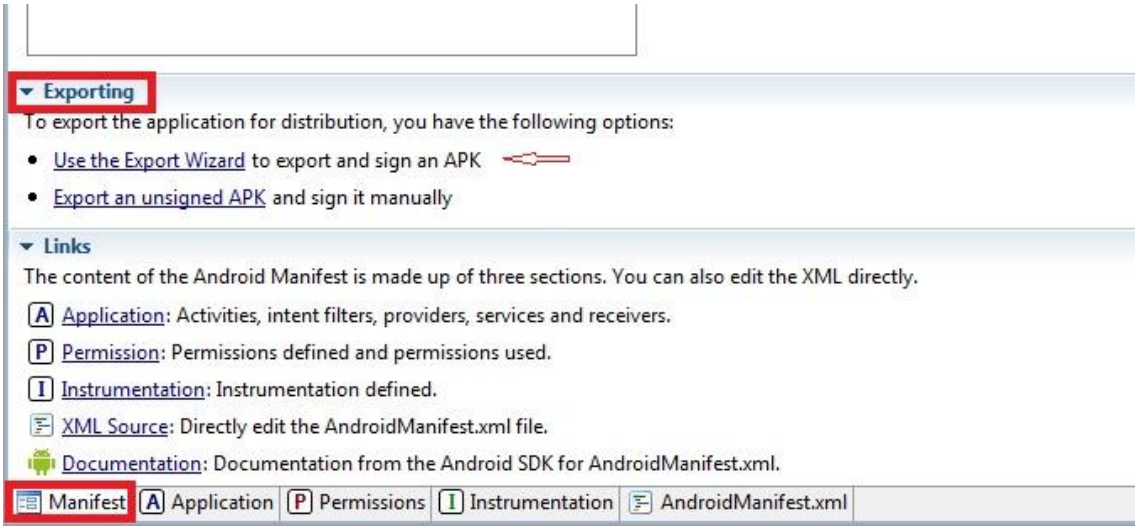


Figura 8. Sección Exporting de Manifiest

Se selecciona la opción Use the Export Wizard.

A continuación, en la siguiente pantalla aparecerá automáticamente el proyecto a firmar, y si no se detecta ningún error, se pulsa en siguiente

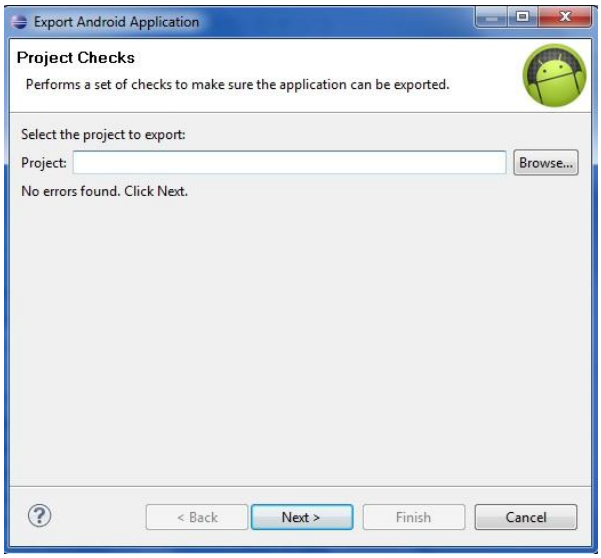


Figura 9. Firma proyecto (1)

Para firmar la aplicación es necesario tener una keystore. Si no se ha creado con anterioridad ninguna, hay que crear una nueva keystore con la opción Create new keystore, rellenando los datos solicitados: Location (directorio donde se guardará la keystore) y password.

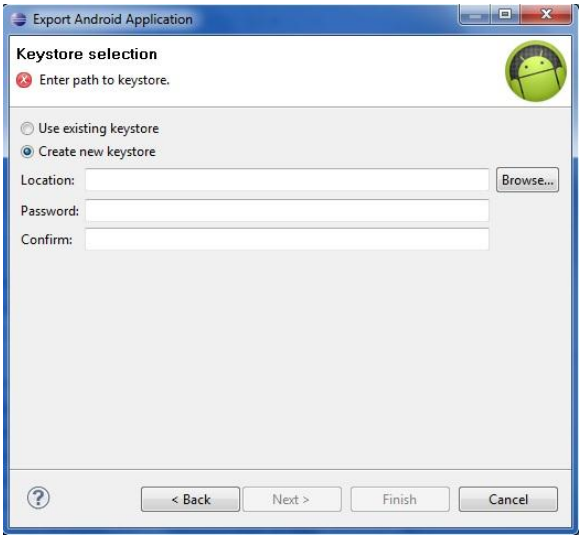


Figura 10. Firma proyecto (2)

En la siguiente pantalla, se rellena el formulario de datos de la keystore y del desarrollador de la aplicación.

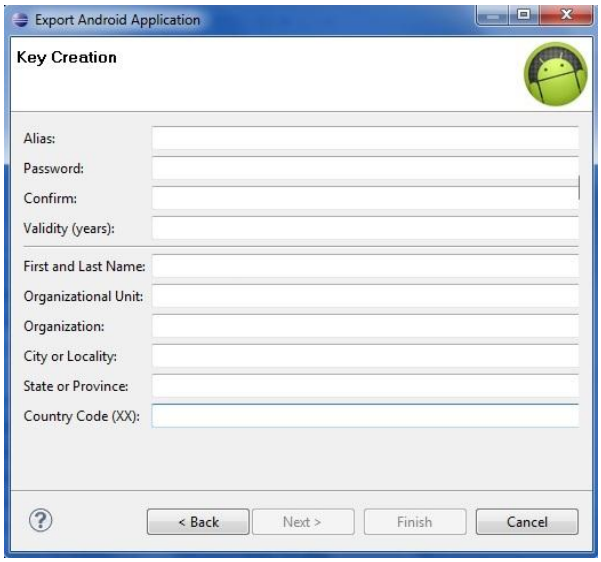


Figura 9. Firma proyecto (3)

Por último, se escoge el directorio en el que se guardara la aplicación ya firmada.

En el caso en el que se quieran firmar más aplicaciones, no será necesario volver a crear una keystore, sino que se podrá utilizar la creada anteriormente, sin necesidad de crear una keystore por aplicación.

ACTUALIZACION

Una vez publicadas las aplicaciones es recomendable actualizarlas corrigiendo errores detectados y añadiéndolas nuevas funcionalidades.

Esto en importante por dos motivos:

- Mantiene la aplicación en las primeras posiciones de la lista de aplicaciones del market, lo que da posibilidad a nuevos usuarios a conocer la aplicación y descargarla.

- Aumenta la confianza los usuarios en el desarrollador, ya que demuestra que tiene en cuenta sus opiniones para mejorar la aplicación.

Para actualizar la aplicación tan solo se debe tener en cuenta que hay que cambiar los atributos android:versionCode y android:versionName.

Cuando se accede al Market con una cuenta de desarrollador, se tienen opciones acerca de las aplicaciones desarrolladas.

Figura 10. Acceso al Market de Android

La Figura 10 muestra dicho acceso. Algunas características y funcionalidades se describen a continuación.

1. **Listado de aplicaciones:** Muestra el número de descargas de cada aplicación, así como el número de descargas activas, que será el número más importante, ya que indica la cantidad de personas que tienen instalada la aplicación (importante si tiene publicidad la aplicación porque aumentará el beneficio).

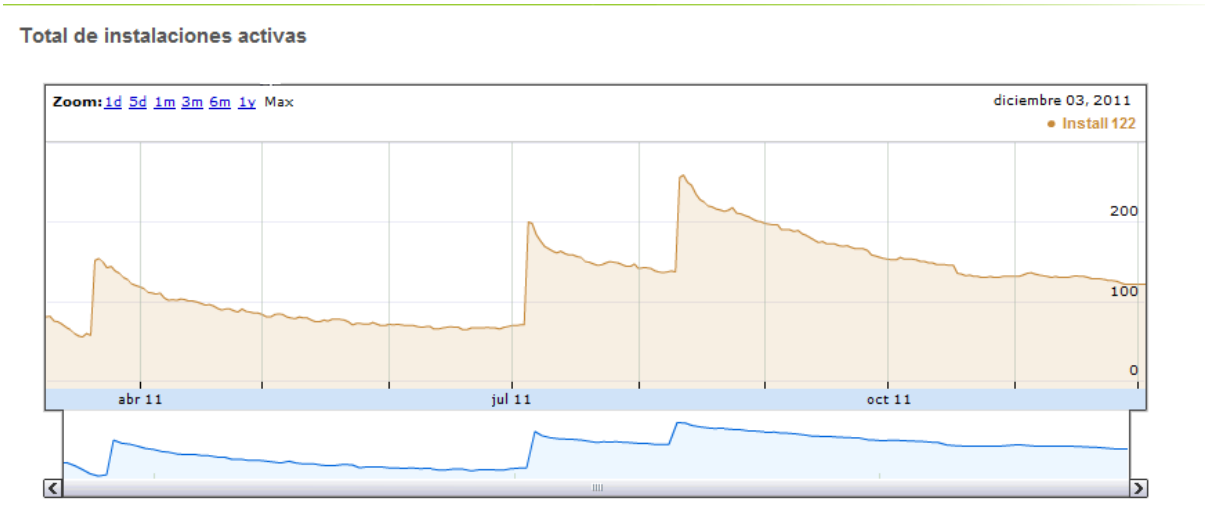


Figura 11. Cargas y descargas sobre las instalaciones activas

En la imagen anterior se observa un gran crecimiento en determinados momentos. Esto es debido a la publicación de actualizaciones de la aplicación, lo que conlleva un aumento del número de descargas al aparecer la aplicación entre las más nuevas, de ahí la importancia de desarrollar actualización de las aplicaciones.

2. **Comentarios:** Para cada aplicación, se puede acceder a las valoraciones y comentarios de los usuarios acerca de esa aplicación.

Figura 12. Comentarios de la aplicación

3. **Errores:** Muestra los errores que se producen en la aplicación y son enviados por los usuarios a los desarrolladores. Generalmente se producen porque determinadas partes de la aplicación no han sido debidamente testeadas, o son errores inesperados.

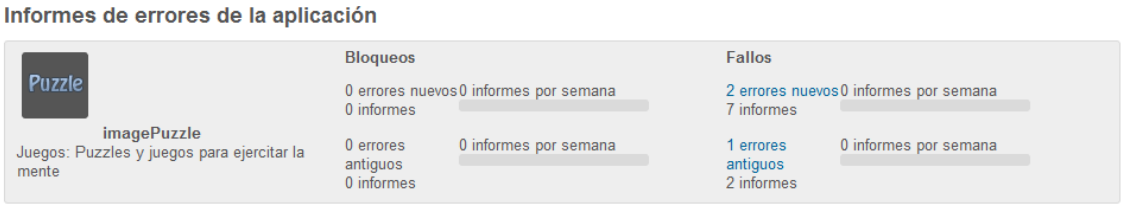


Figura 13. Informe de errores de la aplicación

4. **Disponibilidad de dispositivo:** Indica que dispositivos son compatibles dependiendo del hardware utilizado pudiendo excluir algunos de ellos.

5. **Estadísticas:** Muestra estadísticas actualizadas diariamente de cada aplicación, tales como el número de instancias activas, distribución por versión de Android, por dispositivos, por país o por idioma.

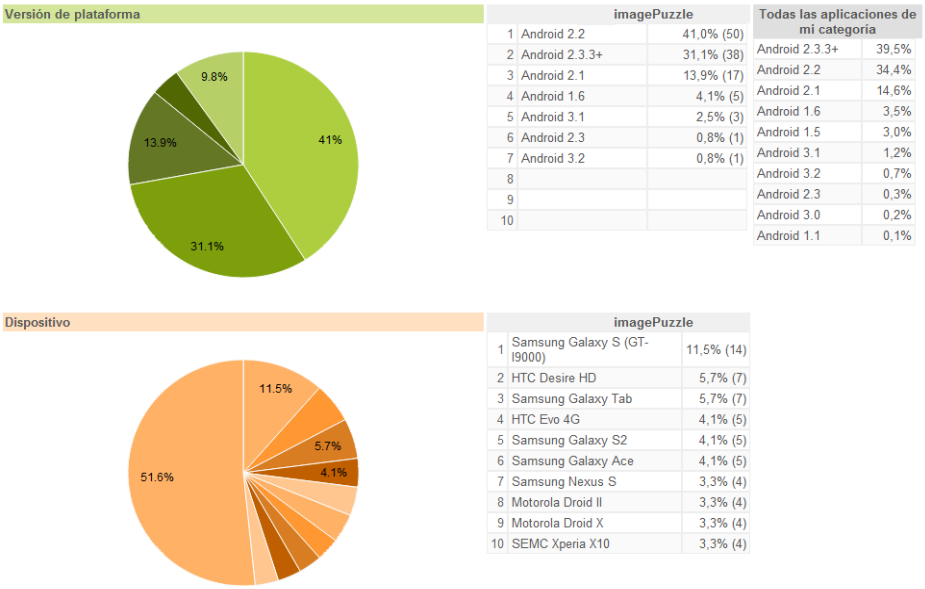


Figura 14. Estadísticas (1)

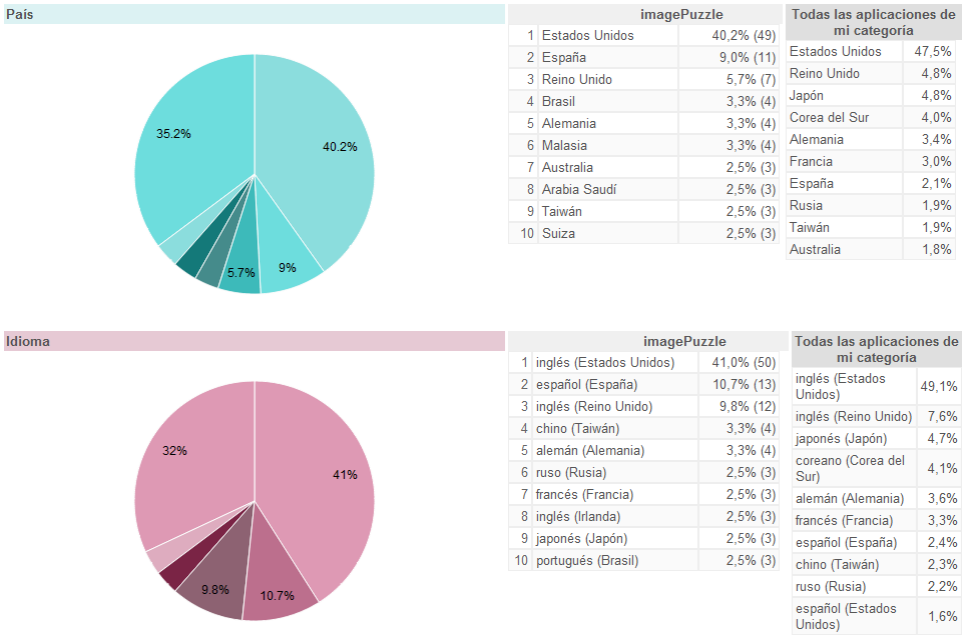


Figura 15. Estadísticas (2)